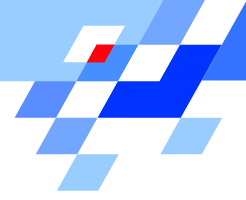
UNIVERSITÄT DORTMUND
Fachbereich Informatik
Lehrstuhl für Software-Technologie



MEMO Nr. 139

Endbericht der Projektgruppe 409: Entwicklung von ortsbasierten Diensten für UMTS-Mobilfunkgeräte (mCube)

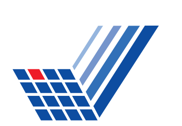
Volker Gruhn Malte Hülder Sami Beydeda (Hrsg.)

Mai 2003

Internes Memorandum des
Lehrstuhls für Software-Technologie
Prof. Dr. Ernst-Erich Doberkat
Fachbereich Informatik
Universität Dortmund
Baroper Straße 301

D-44227 Dortmund

ISSN 0933-7725



Endbericht der Projektgruppe 409 „Entwicklung von ortsbasierten Diensten für UMTS-Mobilfunkgeräte“

Anna Feldgun Tobias Flohre Nils Ganteföhr
Tobias Haltermann Lars Heide Youssef Khlifi
Mario Kleine-Nathland Mika Kossmann Christian Lambert
Miriam Lindhorst Lars Littig Robin Nunkesser
Daniel Pieper Sebastian Ridder

20. Mai 2003

Inhaltsverzeichnis

1 Motivation **1**
1.1 Konzepte . . . 2
1.1.1 Ortsbasierte Dienste (LBS) . . . 2
1.1.2 Online Communitys . . . 2
1.1.3 Mobile Community mit ortsbasierten Diensten . . . 2
1.1.4 Die zusätzlichen Anwendungen . . . 3
1.2 Ziele der Projektgruppe mCube . . . 3
1.3 Die Projektgruppe mCube . . . 4

2 Seminarphase **5**
2.1 Die Vorträge . . . 5
2.2 Formulierung des Projektes . . . 8

3 Übersicht **9**
3.1 Die Community . . . 9
3.2 mCube.dating . . . 9
3.3 mCube.game . . . 10

4 Prozessleitfaden **11**
4.1 Projektplan . . . 11
4.2 Vision . . . 12
4.3 Anforderungsanalyse . . . 12
4.4 Analyse- und Designphase . . . 12
4.5 Prototyp . . . 13
4.6 Weitere Inkremente . . . 13
4.6.1 Das erste Inkrement . . . 14
4.6.2 Das zweite Inkrement . . . 14
4.7 Testen . . . 14
4.8 Zusammenarbeit mit Mobilfunkanbietern . . . 15

5 Qualitätssicherung **17**
5.1 Schulungen . . . 17
5.2 Versionierung . . . 17
5.3 JavaDoc . . . 19
5.4 Code Convention . . . 19
5.5 Assessment Teams . . . 19
5.6 Tests . . . 19

6 Projektplan **21**

7 Anforderungsanalyse **25**
7.1 Namensschema . . . 25
7.2 Anforderungen Community-Basisdienste . . . 25
7.2.1 Funktionale Anforderungen . . . 25
7.2.2 Nicht funktionale Anforderungen . . . 31
7.2.3 Schnittstellenanforderungen . . . 31
7.3 Anforderungen Subsystem mCube.game . . . 32
7.3.1 Voraussetzungen . . . 32
7.3.2 Spielregeln . . . 32
7.3.3 Funktionale Anforderungen . . . 36
7.3.4 Nicht funktionale Anforderungen . . . 38
7.3.5 Schnittstellenanforderungen . . . 39
7.3.6 Auftretende Klassen (vorläufig) . . . 39
7.4 Anforderungen Subsystem mCube.dating . . . 39
7.4.1 Funktionale Anforderungen . . . 40
7.4.2 Nicht funktionale Anforderungen . . . 42
7.5 Anforderungen Subsystem mCube.admin . . . 42
7.5.1 Funktionale Anforderungen . . . 42
7.5.2 Nicht funktionale Anforderungen . . . 44
7.5.3 Schnittstellenanforderungen . . . 44

8 Architektur **45**
8.1 Überblick . . . 46
8.1.1 Architektur . . . 46
8.2 Die Anwendungen . . . 51
8.2.1 mCube.community . . . 51
8.2.2 mCube.dating . . . 55
8.2.3 mCube.game . . . 61
8.2.4 mCube.game.gameEngine . . . 70

9 Technische Infrastruktur **75**
9.1 Application Server . . . 75
9.2 Oracle . . . 75
9.3 Netscape Directory Server . . . 76

10 Entwicklungsumgebung **79**
10.1 Betriebssysteme . . . 79
10.2 Server . . . 80
10.3 Clients . . . 81
10.4 Entwicklungswerkzeuge . . . 81
10.4.1 Sun JDK 1.3.1 und J2EE 1.3 . . . 82
10.4.2 CVS / WinCVS . . . 82
10.4.3 Together 6.0 . . . 82
10.4.4 Bea Weblogic Console/Builder 7.0 . . . 82
10.4.5 Buildprozess mit Ant und CruiseControl . . . 82
10.4.6 Probleme . . . 84

11 Implementierung **87**
11.1 Usermanagement-Komponente . . . 87
11.1.1 Beschreibung . . . 87
11.1.2 Realisierung . . . 88
11.1.3 Probleme . . . 88
11.1.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 89
11.2 Groupmanagement-Komponente . . . 89
11.2.1 Beschreibung . . . 89
11.2.2 Realisierung . . . 89
11.2.3 Probleme . . . 90
11.2.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 91
11.3 Locationing-Komponente . . . 91
11.3.1 Beschreibung . . . 91
11.3.2 Realisierung . . . 91
11.3.3 Probleme . . . 92
11.3.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 93
11.4 Routing-Komponente . . . 93
11.4.1 Beschreibung . . . 93
11.4.2 Realisierung . . . 93
11.4.3 Probleme . . . 95
11.4.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 95
11.5 Matching-Komponente . . . 96
11.5.1 Beschreibung . . . 96
11.5.2 Realisierung . . . 96
11.5.3 Probleme . . . 97
11.5.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 98
11.6 FriendsAround Komponente . . . 98
11.6.1 Beschreibung . . . 98
11.6.2 Realisierung . . . 98
11.6.3 Probleme . . . 100
11.6.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 100
11.7 Communication-Komponente . . . 100
11.7.1 Beschreibung . . . 100
11.7.2 Realisierung . . . 100
11.7.3 Probleme . . . 101
11.7.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 101
11.8 Anwendung mCube.community . . . 101
11.8.1 Beschreibung . . . 101
11.8.2 Realisierung . . . 102
11.8.3 Probleme . . . 102
11.8.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 102
11.9 Anwendung mCube.game . . . 103
11.9.1 Beschreibung . . . 103
11.9.2 Realisierung . . . 103
11.9.3 Probleme . . . 103
11.9.4 Verbesserungs- und Erweiterungsmöglichkeiten . . . 104
11.10 Dating-Komponente . . . 104
11.10.1 Beschreibung . . . 104
11.10.2 Realisierung . . . 104
11.10.3 Probleme . . . 105

12 Installation **107**
12.1 Verwendete Software . . . 107
12.2 Installation des BEA Weblogic Servers . . . 108
12.3 Installation der Oracle Datenbank . . . 108
12.4 Konfiguration des BEA Weblogic Servers . . . 109
12.5 Installation des Netscape Directory Servers . . . 110
12.5.1 Erläuterungen zum LDAP-Server . . . 111
12.6 Externe Java Libraries . . . 113

13 Benutzerhandbuch **115**
13.1 mCube.community . . . 115
13.1.1 Neuanmeldung . . . 115
13.1.2 Login . . . 115
13.1.3 Buddyliste . . . 116
13.1.4 FriendsAround . . . 117
13.1.5 Nachrichten . . . 118
13.1.6 Admin . . . 118
13.1.7 Profile . . . 119
13.1.8 Buddychat . . . 119
13.2 mCube.game . . . 120
13.2.1 Hauptbildschirm und Navigation . . . 120
13.2.2 Erstellen eines neuen Charakters . . . 121
13.2.3 Charakter bearbeiten . . . 123
13.2.4 Nachrichten an andere Spieler . . . 125
13.2.5 Kampf zweier Charaktere . . . 127
13.2.6 Historie eines Charakters . . . 130
13.2.7 Levelaufstieg . . . 130
13.3 mCube.dating . . . 132
13.3.1 Hauptbildschirm und Navigation . . . 132
13.3.2 Das eigene Profil . . . 132
13.3.3 Partner finden . . . 133
13.3.4 Benutzer verwalten . . . 133
13.3.5 Die Mailbox . . . 134
13.3.6 Ein Beispiel: Wie man einen Partner findet. . . . 135

14 Fazit **139**

15 Ausblick **141**

Kapitel 1

Motivation

mCube - dies ist der Name des Produkts, das im Rahmen einer einjährigen Projektgruppenarbeit an der Universität Dortmund entstanden ist. Ziel und Auftrag dieser Projektgruppe war die Entwicklung von ortsabhängigen Diensten für mobile Endgeräte. mCube steht für mobile Community for UMTS-based Entertainment und beschreibt das entstandene Produkt sehr treffend. Es handelt sich um eine Online Community für Mobilfunkgeräte, die auf der Basis und unter Ausschöpfung der Möglichkeiten der UMTS-Technologie ortsabhängige Dienste im Bereich des Entertainments anbieten soll. Die Projektarbeit beschäftigt sich somit mit Ideen und Konzepten für die nahe Zukunft der mobilen Telekommunikation, was dadurch verdeutlicht wird, dass einige Features von mCube mittlerweile von den großen Unternehmen der Branche für den breiten Markt realisiert wurden oder sich in der Planungsphase befinden. Eine besonders wichtige Rolle spielen in diesem Kontext ortsbasierte Dienste, da sie völlig neue Angebote offerieren und zugleich die neuen technischen Möglichkeiten der 3. Generation der mobilen Telekommunikation ausschöpfen. Zu diesen neuen Möglichkeiten zählen vor allem die verbesserte Datenübertragungsrate sowie die viel genauere Ortung von Mobilfunkteilnehmern. Dadurch entsteht eine gegenseitige Abhängigkeitsbeziehung, da zum einen die neue Technik bessere ortsbasierte Dienste ermöglicht und im Gegenzug diese Dienste einen Anreiz zur Nutzung der neuen Technik, mit der immense Kosten verbunden sind, schaffen müssen. Aus dieser Problematik wird ersichtlich, welche entscheidende Rolle Systeme wie mCube spielen und warum sie benötigt werden: Diese Systeme bilden eine Grundlage der zukünftigen mobilen Telekommunikation, weil sie völlig neue Anwendungen generieren und dadurch neue Möglichkeiten schaffen. Von ihrem Erfolg und ihrer Marktakzeptanz hängt deshalb auch zu einem großen Teil der Erfolg der 3. Generation der mobilen Telekommunikation ab. mCube realisiert dabei nicht nur einen dieser erfolgversprechenden, innovativen ortsabhängigen Dienste, sondern erforscht, in welchen verschiedenen Zusammenhängen die Einbindung der Ortsabhängigkeit einen enormen Mehrwert schaffen kann. Aus diesem Grund wurde mit mCube eine Plattform geschaffen, über die der User auf mehrere ortsabhängige Dienste zugreifen kann. Konkret wurden ein ortsbasiertes Spiel (mCube.game) und ein Datingservice (mCube.dating) realisiert und in diese Plattform (mCube.community) integriert. Einen detaillierten Überblick über die Plattform und diese beiden Dienste gibt Kapitel 3. Den zentralen Ausgangspunkt des Projekts mCube bildet ein intelligentes und erfolgversprechendes Konzept, das auf der Zusammenführung verschiedener etablierter Systeme beruht und dadurch einen erheblichen Mehrwert schafft. Dieses Konzept ist der Einstiegspunkt für das Verständnis von mCube und wird somit bereits in diesem Kapitel

erläutert. Insgesamt bietet dieser Endbericht dem Leser die Möglichkeit, die Realisierung von mCube nachzuvollziehen und dabei einen Einblick in wichtige Konzepte und zukunftsweisende Dienste der mobilen Telekommunikation zu gewinnen und gleichzeitig ein System kennenzulernen, das die Grundlage zur Erschaffung dieser Dienste bietet. Der Endbericht beschreibt die Arbeit in der Projektgruppe von der Aneignung von Hintergrundwissen während der einleitenden Seminarphase über die Analyse- und Designphase bis hin zur Implementierung des Projekts. Der Schwerpunkt liegt dabei auf der Entwicklungsphase, weil die teamorientierte Softwareentwicklung eines der Hauptziele der Projektgruppe darstellte. Außerdem steht dem Leser eine Beschreibung der Installation und ein Benutzerhandbuch zur Verfügung. Alles in allem besteht also die Möglichkeit, entweder die komplette Entstehung von mCube Schritt für Schritt nachzuvollziehen oder anhand des Inhaltsverzeichnisses die individuell interessantesten Kapitel auszuwählen, und somit selektiv auf das angebotene Hintergrundwissen zuzugreifen.

1.1 Konzepte

Im Folgenden erläutern wir kurz, welche Konzepte wir zusammengeführt haben, und was unser System enthält und ermöglicht.

1.1.1 Ortsbasierte Dienste (LBS)

Ortsbasierte Dienste sind für mobile Endgeräte konzipiert. Die geografische Position des Anwenders wird benutzt, um bekannte Dienste aufzuwerten oder auch neue anzubieten [Nor01]. Voll genutzt werden können diese Dienste erst mit mobilen Endgeräten, die eine genaue Ortung erlauben. Im Zuge der bevorstehenden Einführung von UMTS[1] wird die Ortung zunehmend genauer [KAL$^+$01]. Aber auch mit der bisherigen Ortungsgenauigkeit sind schon viele ortsbasierte Dienste möglich.

1.1.2 Online Communitys

Eine Community ist durch eine many-to-many Interaktion von Benutzern mit gemeinsamen Interessen ausgezeichnet. Eine Online Community bietet diesen Benutzern größtenteils computergestützte Interaktionsmöglichkeiten wie Chatrooms, Message Boards, E-Mails und Sofortnachrichten [Int01]. Zusätzlich gibt es noch Suchmöglichkeiten nach Benutzern mit gleichen oder ähnlichen Interessen. Dadurch sind die Benutzer in der Lage, andere Benutzer mit gleichen Interessen zu finden und mit ihnen zu interagieren.

1.1.3 Mobile Community mit ortsbasierten Diensten

Unser Ansatz ist es nun, die Vorzüge von Online Communitys und ortsbasierten Diensten miteinander zu verbinden. Es gibt viele ortsbasierte Dienste, die gerade in many-to-many Interaktionen Sinn haben. Dazu zählt zum Beispiel die Anzeige befreundeter Personen in geografischer Nähe. Andersherum ist es ein Mehrwert für eine Online Community, auf mobilen Endgeräten Ortsinformationen bieten zu können. Zudem bietet eine Community eine optimale Plattform für ortsbasierte Dienste. mCube.community bietet dem Benutzer die Grundfunktionalitäten einer Community sowie Ortung von

[1] Universal Mobile Telecommunications System

befreundeten Personen, Eingabe und Abgleich von Profilen, Kommunikation und ortsbasierte Interaktion. Durch den komponentenbasierten Aufbau wurden die Voraussetzungen geschaffen, damit die Community als Plattform für Anwendungen dienen kann. Diese Anwendungen können dann auch von Fremdanbietern stammen und die Basisdienste nutzen.

1.1.4 Die zusätzlichen Anwendungen

mCube.dating Die Dating-Anwendung basiert auf den bekannten und akzeptierten Eigenschaften eines Dating-Services und erweitert diese um innovative Features. Die größte Neuerung ist dabei sicherlich die Ortsabhängigkeit, die eine völlig neue Variante des Datings erschafft. Es ist mittels mCube.dating möglich, kontaktfreudige Personen, deren Persönlichkeitsprofile mit dem des Anwenders zusammenpassen, innerhalb einer gewissen Umgebung zu lokalisieren und daraufhin zu kontaktieren. Somit ist der Anwender in der Lage, zu jeder Zeit und an jedem Ort die Person zu treffen, die sowohl charakterlich als auch vom äußeren Erscheinungsbild seinen Wünschen und Vorstellungen entspricht. Dieses Konzept schafft eine sehr gute Basis, um jemanden näher kennen zu lernen, und könnte maßgeblich zum Erfolg von mCube.dating beitragen.

mCube.game Diese Spiel fügt die erfolgreichen Konzepte der Sammelkartenspiele und der Kurzkampfspiele (Lycos Prügelpause) zu einer innovativen und attraktiven Anwendung zusammen [Lyc02, Bro97]. Basisidee ist ein Rundenkampf, der von erweiterbaren Charakteren, zunächst ortsunabhängig, zwischen zwei Handybesitzern ausgetragen wird. Die Charaktere werden durch sammelbare elektronische Karten um Funktionalität und Stärke erweitert. Diese sollen in Aussehen und Auflösung mit Handylogos vergleichbar sein. Das Sammeln von Logos und Klingeltönen für Handys und der bisherige Erfolg von Sammelkartenspielen sind Vorbilder für den erhofften Erfolg des Spielkonzepts. Für interessierte Firmen bietet sich die Möglichkeit Karten im Rahmen von Werbeaktionen zu verschenken.

1.2 Ziele der Projektgruppe mCube

Die Projektgruppe „mCube" verfolgte mehrere Ziele. Da es sich bei der Projektgruppe um eine Lehrveranstaltung handelt, war der Erfahrungsgewinn das wichtigste Ziel. Dieser fand auf mehreren Bereichen statt. Eine der wichtigsten neuen Erfahrungen war die Konzeption und Realisierung eines größeren Softwareprojekts. Das praktische Erlernen des Softwareentwicklungsprozesses XPM mit inkrementellen Lebenszyklusmodell ([BT75]) und die Herausbildung der Teamfähigkeit sind hier die wichtigsten Aspekte. Des Weiteren sollte auch ordentliche Dokumentation und Qualitätssicherung erlernt werden. Da Wirtschaftskontakte für jede Projektgruppe sinnvoll (für unsere sogar unabdingbar) sind, waren diese auch ein Ziel der Projektgruppe. Die Kontakte an sich waren gut zu knüpfen und mehrere Unternehmen signalisierten Kooperationsbereitschaft. Nachdem die Entscheidung zur Kooperation mit T-Mobile getroffen wurde, entstand diese leider zu spät für einen Nutzungsumfang, der ausgereicht hätte, um alle konzeptionellen Ziele zu verwirklichen.

1.3 Die Projektgruppe mCube

Die Organisation einer Projektgruppe soll in etwa der Organisation von Projekten in der Wirtschaft entsprechen. Zur Erreichung dieses Zieles wurden innerhalb der Projektgruppe Rollen verteilt. Im Gegensatz zur Wirtschaft konnte die Rollenverteilung jedoch nicht hauptsächlich auf Kompetenz beruhen. Bei unserer Rollenverteilung kam zusätzlich noch Interesse und Engagement hinzu, da es für manche Rollen schlichtweg keine Experten gab. Die festen und dauerhaften Rollen sind „Projektmanagerin“ (Miriam Lindhorst), „System-Administrator“ (Nils Ganteföhr, Lars Heide), „Qualitätsmanager“ (Mario Kleine-Nathland), „Marketingexpertin“ (Anna Feldgun) und „Architekt“ (Tobias Flohre, Sebastian Ridder, Daniel Pieper, Christian Lambert). Vorübergehende Rollen waren „UMTS-Experte“ (Mika Kossmann, Lars Littig, Youssef Khlifi) und „GUI-Designer“ (Tobias Haltermann, Mario Kleine-Nathland). Hinzu kommt die Rolle „Entwickler“, die prinzipiell jeder inne hat.
Neben diesen Rollen wurden meist Kleingruppen bzw. Kleinstgruppen gebildet, die Arbeiten übernommen haben.

Kapitel 2

Seminarphase

Zu Beginn des Projektes wurde eine Seminarphase durchgeführt, innerhalb derer je zwei Personen einen Vortrag zu einem vorgegebenen Thema ausgearbeitet und während eines gemeinsamen Seminarwochendes vorgestellt haben. Die Seminarphase wurde zusammen mit der PG 411 durchgeführt, die sich mit dem Thema „E-Bill-Presentment“ befasst.
Ziel der Seminarphase war es, für den weiteren Verlauf der PG wichtige Themen zu erarbeiten und die Teilnehmer auf einen gemeinsamen Wissensstand zu bringen. Darüber hinaus erfolgte die erste konkrete Formulierung unserer Projektidee und eine Aufteilung der Teilnehmer in Rollen.

2.1 Die Vorträge

Die Themenwahl orientierte sich an den spezifischen Fragestellungen, die für die Arbeit der beiden teilnehmenden Projektgruppen zu erwarten waren. Die Vorträge gaben allen Teilnehmern einen ersten Einblick und ermöglichten so erste Überlegungen für spätere Architekturentscheidungen. Zudem wurde Fachwissen vermittelt. Die Vorträge unterteilten sich in die Bereiche „Softwaretechnische Aspekte“, „Technische Grundlagen“ und „Wirtschaftliche Rahmenbedingungen“. Im Bereich „Softwaretechnische Aspekte“ wurden Vorträge zu den Themen

- „Softwareentwicklungsprozesse“
- „Komponentenmodelle“
- „Konfigurationsverwaltung und Projektdokumentation“ und
- „Benutzeroberflächen“

gehalten.
Der Bereich „Technische Grundlagen“ war durch

- „Applikationsserver“
- „XML-basierte Datenaustauschformate“
- „Telematik-, Mobilfunk- und Netzwerktechniken“

- „Sicherheit“
- „UMTS“ und
- „Location Based Services“

vertreten.
Zum Thema „Wirtschaftliche Rahmenbedingungen“ hörten wir Referate mit den Titeln

- „E-Commerce / E-Business“ und
- „Mobile Commerce“.

Die Folien der Vorträge sowie die ausführlichen schriftlichen Ausarbeitungen sind in elektronischer Form auf der Homepage der Projektgruppe (www.mcu.be) verfügbar. Im Folgenden werden die Inhalte der Vorträge kurz beschrieben.

Softwareentwicklungsprozesse Nach einer kurzen Begriffserläuterung wurden einige Prozessmodelle vorgestellt und auf ihre Tauglichkeit bezüglich ihrer Verwendung in einer PG bewertet. Prototyping und das Spiralmodell sind dabei als Möglichkeiten vorgeschlagen worden. XtremeProgramming war in der begleitenden Diskussion sehr umstritten. Im weiteren Verlauf vorgestellte Qualitätsmodelle wurden als zu komplex eingestuft. Bei der Vorstellung der Notationssprachen UML, Petrinetze und EPK sind UML und Petrinetze als anwendbar eingestuft worden.

Komponentenmodelle Der Vortrag erklärte die Vorteile komponentenbasierter Softwareentwicklung und stellte die gängigsten Komponentenmodelle detailliert vor. Der Schwerpunkt lag dabei klar auf Microsofts Lösung .NET und der Java 2 Enterprise Edition, die von Sun vertrieben wird.
Die Vorteile beim Einsatz von Komponentenmodellen liegen in der Unterstützung der Punkte Persistenz, Skalierbarkeit und Transaktionsmanagement. Auf der anderen Seite sind fast alle Lösungen plattform- und/oder herstellerspezifisch.
Es wurde die Empfehlung ausgesprochen, entweder Microsoft .NET oder J2EE zu verwenden.

Konfigurationsverwaltung und Projektdokumentation Die Notwendigkeit eines Konfigurationsmanagements in modernen Softwareentwicklungsprozessen wurde motiviert. Nach einer Einführung in die grundlegende Funktionsweise vorhandener Tools wurde eine Marktübersicht über die verfügbaren Tools gegeben, darunter PVCS, CVS, VSS und ClearCase. Die Produkte wurden mit ihren Vor- und Nachteilen ausführlich vorgestellt.
Als Fazit wurde entschieden empfohlen, das System CVS einzusetzen, da es frei verfügbar und zuverlässig sei.

Benutzeroberflächen Die Grundlagen zur Gestaltung von Benutzeroberflächen und deren Bedeutung für die Akzeptanz eines Softwareproduktes wurden erläutert. Am Beispiel des Dialogdesigns wurden diese abstrakten Regeln erläutert. Dabei wurden die Probleme beim Einsatz von mobilen Endgeräten gesondert betrachtet.
Neben einer Darstellung des Entwicklungsprozesses für grafische Oberflächen wurde auch kurz auf das Testen der Usability eingegangen.

Applikationsserver Zunächst wurde die historische Entwicklung und die heutige Notwendigkeit von mehrschichtigen Architekturen erläutert. Die einzelnen Schichten wurden einzeln vorgestellt. Danach wurde die Technologie von Applikationsservern beschrieben. Am Beispiel eines J2EE konformen Servers wurden die Möglichkeiten und der Aufbau von Applikationsservern dargestellt.
Als Fazit wurden die Vorzüge beim Einsatz eines Applikationsservers herausgestellt, es wurde aber auch vor einem hohen Einarbeitungsaufwand gewarnt. Eine Empfehlung für einen konkreten Server wurde nicht abgegeben.

XML-basierte Datenaustauschformate Nach einer kurzen Einführung in den Aufbau von XML wurde vor allem auf dessen Bedeutung für den Austausch von Daten und das Thema Web-Services eingegangen. Da insbesondere Lösungen im Banken- und im E-Commerce-Umfeld beschrieben wurden, war der Vortrag für die Arbeit unserer Projektgruppe weniger interessant.

Telematik-, Mobilfunk- und Netzwerktechniken Der Vortrag gliederte sich in die drei recht unabhängigen Bereiche Telematik, Mobilfunktechnik und Netzwerktechnik. Im Bereich Telematik wurden nach einer ausführlichen Definition mögliche Einsatzbereiche näher erläutert. Als Abschluss wurden einige bereits vorhandene System kurz vorgestellt.
Der Bereich Mobilfunktechnik erklärte die Funtionsweise von Mobilfunknetzen und erzählte die historische Entwicklung mobiler Telefonie.
Im Netzwerkbereich wurden die Modelle ISO/OSI und TCP/IP erläutert. Danach wurden Netzwerktopologien und -technologien beschrieben.

Sicherheit Im Vortrag über Sicherheit wurde der Unterschied symmetrischer und asymmetrischer Verschlüsselungsverfahren erklärt. Danach wurden aus jedem Bereich implementierte Verfahren vorgestellt. Als Fazit wurde der Einsatz asymmetrischer Verfahren als notwendig erachtet. Implementierungen sind im aktuellen Java Development Kit der Firma Sun vorhanden.

UMTS Der Vortrag gab einen ausführlichen Überblick über die Technik und die Architektur von UMTS. Nach einer Motivation für die Einführung einer neuen Mobilfunkgeneration wurde die Architektur des UMTS-Netzes und die neue Übertragungstechnik erklärt. Zum Abschluss wurde die neuartige Dienstarchitektur beschrieben.

Location Based Services Nach einer Marktübersicht und einer kurzen Marktstudie wurde als Schwerpunkt die technische Umsetzung von Location Based Services im UMTS-Netzwerk beschrieben. Hierbei wurde auch eine Plattform namens miAware erwähnt, in der viele der von uns benötigten Dienste laut Hersteller bereits implementiert und komfortabel nutzbar sind. Auch auf die gesellschaftlichen Nutzen und Bedenken von Location Based Services wurde eingegangen.

E-Commerce / E-Business Nach einer Definition wurden mehrere Ansätze zur Klassifizierung von E-Business vorgestellt: die Klassifizierung nach Phasen einer Handelstransaktion und die Klassifizierung nach Art der Akteure. Diese wurden an Beispielen erläutert. Im Anschluss wurden mögliche Architekturen zur Realisierung von E-Business vorgestellt.

2.2 Formulierung des Projektes

Der PG-Antrag legte das Projekt noch nicht auf eine konkrete Anforderung fest, sondern forderte lediglich die Umsetzung eines Location Based Services. Im Laufe der Diskussion festigte sich die Vorstellung, eine Community aufzubauen, die um Anwendungen ergänzt werden kann. Als Referenzanwendungen sollte auf jeden Fall ein Dating Service implementiert werden.
Darüber hinaus wurde über ein Spiel nachgedacht. Nachdem in der Seminarphase kein Ergebnis feststand, wurde dieses nach der Seminarphase entworfen. Weiterhin wurde beschlossen, eine „Vision" unseres Projektes zu erarbeiten, was nach der Seminarphase geschah.

Kapitel 3

Übersicht

Das folgende Kapitel gibt eine Übersicht auf das mCube-System aus der Vogelperspektive.
Die einzelnen Module der Community werden kurz vorgestellt, die darunter liegenden technischen Aspekte jedoch den weiteren Kapiteln vorbehalten.

3.1 Die Community

Basis des mCube-Systems ist die Plattform mCube.community.
Diese bietet dem Benutzer die Grundfunktionalitäten einer Internet-Community (also Treffpunkt, Informationsaustausch etc.) sowie die Möglichkeit der Ortung befreundeter Personen und andere ortsbasierte (Location Based) Services.
Bedingung zur Teilnahme an der Community ist die erfolgreiche Anmeldung unter Angabe persönlicher Daten in Profilform. Danach stehen alle Services in Form eines Internetportals zur Verfügung.
Beispiele für exponierte Basisfunktionen, die den Mitgliedern, die Erlaubnis der beteiligten Benutzer vorausgesetzt, zur Verfügung stehen, sind die Ortsanzeige befreundeter Personen (FriendsAround) sowie die Teilnahme an Chats.

3.2 mCube.dating

Eines von zwei auf der Plattform aufsetzenden Modulen, die von uns zur Demonstration der Funktionsweise des Systems realisiert wurden, ist das Dating Modul.
Hinter diesem Namen verbirgt sich ein 3G Dating-Service der mobilen Telekommunikation. In Zeiten, in denen die Zahl der Singles kontinuierlich wächst, und sich bisherige, Internet basierte Flirtchats und Dating-Services immer größerer Beliebtheit erfreuen, wird diese Applikation auf breite Akzeptanz am Markt treffen. Die größte Neuerung ist dabei die Ortsbasiertheit, die eine völlig neue Variante des Datings ermöglicht. Mittels mCube.Dating ist es möglich, kontaktfreudige Personen, deren Profile mit dem des suchenden Benutzers zusammenpassen, innerhalb einer gewissen Umgebung zu lokalisieren und daraufhin zu kontaktieren. Somit wird der Benutzer in die Lage versetzt, zu jeder Zeit und an jedem Ort die Person zu treffen, die sowohl charakterlich als auch von den Interessen seinen Wünschen und Vorstellungen entspricht.
Im Detail sieht die Benutzung des Services wie folgt aus:

Der Anwender hat zunächst die Möglichkeit, sein Profil über sein (mobiles) Endgerät zu erstellen. Das Profil gliedert sich dabei in Informationen bezüglich des äußeren Erscheinungsbildes, des Charakters, der Hobbys plus weiterer persönlicher Daten.
Ebenfalls können hier Wünsche und Vorstellungen bezüglich des gesuchten Partners gespeichert werden. Ist das Profil erst einmal erstellt, kann der Benutzer jederzeit den Service benutzen und somit herausfinden, ob sich Personen, die seinen Vorstellungen entsprechen, innerhalb eines definierten Radius um seinen momentanen Aufenthaltsort befinden. Dabei wird der Benutzer über sein Mobiltelefon geortet, und es erfolgt ein Abgleich der Profile des Benutzers mit denen der übrigen Community-Mitglieder, die ebenfalls am .dating Service teilnehmen. Hierbei besteht die Möglichkeit festzulegen, wann man von anderen Benutzern lokalisiert werden kann und wann nicht.
Ist die gefundene Person ebenfalls an einem Treffen interessiert, haben nun beide die Wahl, ob sie sich auf ein Blind-Date einlassen wollen oder ob sie vorher mittels der Kommunikationsfunktionalitäten der Communtiy Kontakt aufnehmen wollen.

3.3 mCube.game

Das zweite Modul ist ein Spiel, das ortsbasierte Werbeaktionen und ortsbasiertes Spielen unterstützt.
Idee hierbei war es, mehrere erfolgreiche Konzepte zusammenzuführen: Sammelkarten Spiele (Trading Card Games) wie z.B. Pokemon o. Ä., Kurzkampfspiele (Lycos Prügelpause) und die Ortsbasiertheit. Grundlage ist der Rundenkampf, der von erweiterbaren Charakteren zunächst ortsunabhängig ausgetragen wird. Die Charaktere werden durch sammelbare elektronische Karten um Funktionalität und Stärke erweitert. Diese sind in Aussehen und Auflösung mit Handylogos vergleichbar. Die Karten werden sowohl ortsunabhängig kostenpflichtig (Internet) als auch in Kooperationen mit interessierten Unternehmen ortsabhängig (also z.B. in einer Filiale des Unternehmens) kostenlos vertrieben.
Das Spiel selbst basiert auf Charakteren mit den üblichen Basiseigenschaften: Stärke, Geschicklichkeit und Intelligenz sowie Konstitution. Die Karten können nun verschiedene Funktionen zur Erweiterung des Charakters übernehmen. Es gibt Kartentypen, die zur Steigerung der Basiseigenschaften dienen, sowie Waffenkarten und Karten mit Spezialaktionen. Basis für eine Kampfrunde bilden die Eigenschaften des Charakters in Zusammenhang mit der eingesetzten Waffe. Nachdem jeder Spieler seine Grundaktion gewählt hat, wird eine Zufallsentscheidung getroffen, die den Stärkeren bevorzugt. Beendet ist der Kampf, wenn ein Charakter keine Hitpoints mehr hat. Der Charakter kann aber noch weiterhin benutzt werden.

Kapitel 4

Prozessleitfaden

Als eine wesentliche Ausprägung unserer wissenschaftlichen Arbeitsweise bei dem Projekt haben wir uns entschlossen, die von uns verwendeten Prozessmuster genau zu dokumentieren.

Als Prozessmodell haben wir hauptsächlich das evolutionäre Prototyping benutzt. Aufbauend auf dem ersten Prototypen haben wir weitere inkrementelle Phasen durchlaufen, in denen sukzessive immer mehr jeweils zu Beginn einer Phase festgelegte Funktionalitäten implementiert wurden. Die einzelnen Besonderheiten unseres Prozessverlaufs werden im Folgenden näher erläutert.

4.1 Projektplan

Um einen genauen Zeitplan festzulegen und um jederzeit unsere Fortschritte daran überprüfen zu können, haben wir einen detaillierten Projektplan in der Form eines Gantt-Charts aufgestellt. Gantt-Charts wurden bereits 1917 entwickelt [MM00], sind aber auch heute noch eine der nützlichsten Methoden, um einen Zeitplan zu präsentieren. Sie zeigen Aufgaben und zeitliche Vorgaben sowie zusätzliche Informationen wie z.B. welche Aufgaben Meilensteine darstellen oder welche Ressourcen (PG-Teilnehmer) jeweils gebraucht werden.

Grundsätzlich ist einem Gantt-Chart nicht zu entnehmen, wie die einzelnen Aufgaben zusammenhängen [LR02]. Durch die Benutzung von Microsoft Project hatten wir aber auch diese zusätzliche Funktionalität zur Verfügung; es wurden Pfeile in den Zeitplan eingefügt, denen man z.B. entnehmen konnte, dass eine Aufgabe so stark von einer anderen abhing, dass sie erst begonnen werden konnte, wenn die andere Aufgabe bereits abgeschlossen war.

Der Zeitplan wurde erst recht grob über den gesamten Zeitraum der Projektarbeit erstellt und in weiterer Arbeit immer mehr konkretisiert und verfeinert, so dass immer ein detaillierter Plan über die in näherer Zukunft zu erledigenden Aufgaben vorhanden war. Die jeweils aktuelle Version war immer online für alle PG-Teilnehmer verfügbar, damit sich jeder jederzeit einen Überblick über den Projektstand und die noch zu erledigenden Aufgaben machen konnte.

4.2 Vision

Am Anfang unseres Projektes stand die Entwicklung einer „Vision“. In einem gemeinsamen Brainstorming haben wir die Grundideen der zu entwickelnden Anwendung(en) im Groben festgelegt, ohne näher auf die einzelnen Funktionen einzugehen. Hierbei stand auch die Ausrichtung auf den Marketinggedanken im Vordergrund, da wir dieses Dokument den Mobilfunkbetreibern zur Verfügung gestellt haben, um sie von den Vorteilen einer Zusammenarbeit zu überzeugen.

4.3 Anforderungsanalyse

Anschließend haben wir ein umfassendes Anforderungsdokument erstellt. Unter mehrmaliger Erweiterung haben wir das Projekt grob in die funktionalen Teilbereiche Community, Game, Dating und Administrationsdienste eingeteilt und dann im Einzelnen alle gewünschten Funktionen festgelegt. Dabei wurde für jede Funktion entschieden, inwieweit es sich dabei um eine MUSS-Anforderung handelt, die auf jeden Fall implementiert werden muss, oder um eine KANN-Anforderung, die nur realisiert werden soll, wenn die zwingenden Funktionen bereits alle funktionieren.

4.4 Analyse- und Designphase

In der Analysephase wollten wir zunächst das klassische objektorientierte Analysemodell verwenden, wobei wir allerdings auf Schwierigkeiten gestoßen sind. Durch unsere grundsätzliche architekturelle Entscheidung, EJB 2.0 zu verwenden, die von der Spezifikation her Vererbung nur unzureichend unterstützen, und aufgrund der Tatsache, dass unsere Teilbereiche wenige Überschneidungen haben, haben wir uns von der klassischen Modellierung mit der Vererbung einzelner Java Klassen stark entfernt, so dass ein übliches UML-Klassendiagramm (in dem auch die Vererbungsbeziehungen erfasst sind) nicht ganz sachgerecht erschien.
Wir haben daraufhin mit einer Aufteilung unseres Projektes in logische Zusammenhänge begonnen und diese Zusammenhänge dann direkt in Komponenten umgesetzt. Dies war eine direkte Weiterentwicklung aus der groben Unterscheidung der funktionalen Teilbereiche in der Anforderungsanalyse, die dadurch teilweise abgeändert wurden. So wurden z.B. die Admin-Dienste in die Community mit eingegliedert. Es handelte sich hierbei um eine hauptsächlich architekturelle Aufgabe, also haben wir das Ergebnis in einem sog. Architekturdokument zusammengefasst.
Die Komponenten, die sich dabei herausgebildet haben, sind unterschieden in die Komponenten der core-Schicht und die eigentlichen Anwendungen.
Komponenten der core-Schicht sind die Folgenden:

- Usermanagement
- Gruppenmanagement
- Locationing
- Routing
- FriendsAround
- Matching

- Communication

Wir hatten zunächst noch eine Logging-Komponente geplant, haben dann jedoch den Log4j Logger aus dem Apache Jakarta Projekt benutzt, der alle von uns geforderten Funktionalitäten besitzt. Log4j ermöglicht, mit Hilfe von XML-Konfigurationsdateien, eine Änderung des Loglevels während der Laufzeit, so dass große Geschwindigkeitseinbußen vermieden werden können, da die log-Statements auch im endgültigen Code erhalten bleiben und nur nach Bedarf ein- oder abgeschaltet werden können. Außerdem bietet Log4j eine Vererbungshierarchie für Logger, was ebenfalls die Performanzkosten minimiert, da so fein granular bestimmt werden kann, für welche Komponenten die log-Statements ausgewertet werden sollen.
Die Anwendung ist aufgeteilt in:

- Community
- Dating
- Game

Für jede dieser Komponenten haben wir dann im Architekturdokument nach einer kurzen Beschreibung im Einzelnen festgelegt, welche Anforderungen an die Komponente gestellt werden, und einen möglichen internen Aufbau dargelegt. Dabei haben wir bereits eine Übersicht über die Funktionen der jeweiligen Komponente im Sinne einer API entwickelt und auch UML-Klassendiagramme für jede einzelne Komponente angelegt.
Auch dieses Architekturdokument hat mehrere Entwicklungsstufen durchlaufen, in denen es immer detaillierter wurde. In der Version 2.0 war es soweit fertig, dass es als Grundlage für den Prototypen dienen konnte.

4.5 Prototyp

Anschließend haben wir ausgehend von dem Architekturdokument mit der Implementierung eines Durchstichprototypen begonnen. Dafür haben wir uns in Kleingruppen aufgeteilt, die jeweils für einzelne Komponenten verantwortlich waren. Dieser Prototyp sollte einen Cut-Through durch alle Ebenen realisieren und die Kommunikation zwischen allen Komponenten testen.
Dabei hat das Architekturdokument fortwährend Änderungen erfahren, die allerdings nur auf der Ebene der verwendeten Funktionen stattgefunden haben. Sehr hilfreich war dafür die starke Kommunikation unter allen Kleingruppen, die durch die gemeinsame Entwicklung im PG-Pool möglich war. Mit dem positiven Ergebnis, dass der Prototyp unseren Erwartungen weitestgehend entspricht, haben wir auch im Nachhinein das Funktionieren unserer Herangehensweise in der Analyse- und Designphase gezeigt. Mit diesem Prototypen haben wir das erste Semester der Projektgruppe abgeschlossen.

4.6 Weitere Inkremente

Die weitere Entwicklung unseres Projektes erfolgte in zwei Inkrementen. Die einzelnen Anforderungen für jedes Inkrement wurden jeweils zu Beginn einer neuen Phase, ausgehend von dem bisher erreichten Stand, festgelegt.

4.6.1 Das erste Inkrement

Das erste Inkrement setzte direkt auf dem Prototypen auf. Hier ging es darum, die grundlegenden Funktionen vor allem der core-Komponenten zu verbessern. Insbesondere wurden das User- sowie das Gruppenmanagement verfeinert, und es wurde ein Matching-Algorithmus erarbeitet.

An dieses Inkrement hat sich noch eine kurze Nachbesserungsphase angeschlossen, bis wir gegen Mitte des zweiten Semesters das erste Inkrement präsentieren konnten.

4.6.2 Das zweite Inkrement

Einer der wichtigsten Punkte des zweiten Inkrementes war die Anpassung der GUI an die unterschiedlichen Anforderungen, je nachdem, ob man die Anwendungen mit einem normalen oder einem Handybrowser aufruft. Hierbei haben wir uns für den Einsatz von XML und XSLT entschieden, was eine gewisse Einarbeitungszeit erforderlich machte. Zuerst wollten wir das Apache Cocoon Framework benutzen. Dieses erschien dann aber doch als zu umfangreich. Schließlich haben wir uns für die Benutzung von Filtern (Java-Servlet-Spezifikation 2.3) entschieden.

Des Weiteren gab es einige funktionale Verbesserungen bei den Anwendungen Community, Game und Dating.

Für die core-Komponenten ging es darum, die Anbindung an die von T-Mobile bereitgestellte Location Based Services API zu realisieren. Dafür wurde eine neue Klasse entwickelt, die sich mit einer SSL-gesicherten http-Verbindung mit dem T-Mobile-Server verbinden kann, um die Lokalisierungsdaten von dort abzurufen.

Wir haben die Implementierung dann mit dem vollendeten zweiten Inkrement abgeschlossen, obwohl zunächst drei Inkremente geplant waren. Der Grund dafür ist, dass die Komplexität der einzelnen Inkremente, an die sich immer auch noch eine Verbesserungsphase angeschlossen hat, größer war, als es bei drei Inkrementen realisierbar gewesen wäre. Eine Einteilung in Inkremente geringerer Komplexität hätte dazu geführt, dass nicht alle Teilnehmer immer etwas zu jedem Inkrement hätten beitragen können, und wäre damit dem Charakter der Projektgruppe als Lehrveranstaltung nicht gerecht geworden.

4.7 Testen

An die Implementierung hat sich eine umfangreiche Testphase angeschlossen.

Für die Anwendungen wurden Testlisten erstellt, in denen jeweils die Aktion und das gewünschte Sollergebnis eingetragen waren. Dann haben jeweils Personen, die nicht in die jeweilige Implementierung involviert waren, die einzelnen Testfälle durchgespielt und das jeweilige Ergebnis in die Testlisten eingetragen. Daraufhin wurden - soweit nötig - Verbesserungen vorgenommen, bis alle Testfälle erwartungsgemäß verliefen.

Die core-Komponenten wurden mit Hilfe von JUnit-Tests getestet. JUnit ist ein automatisches Test-Framework für Java Klassen. Es werden Testfälle mit erwarteten Abläufen erstellt, und das Verhalten der getesteten Klasse wird dann mit dem erwarteten Verhalten verglichen.

4.8 Zusammenarbeit mit Mobilfunkanbietern

Wir haben uns von Beginn der Projektgruppen an sehr intensiv darum bemüht, einen Partner aus der Mobilfunkbranche zu finden, da wie einige unserer Ideen nicht ohne fremde Hilfe realisieren konnten. Im Laufe des ersten Semesters haben wir mit mehreren Mobilfunkunternehmen Kontakt gehabt, bis wir uns für eine Zusammenarbeit mit T-Mobile entschieden hatten.
In den Gesprächen mit allen Mobilfunkbetreibern stellte sich schnell heraus, dass die von uns anvisierte UMTS-Technik noch nicht so weit fortgeschritten ist, dass wir darauf aufbauen könnten. Im Laufe dieser Gespräche hat sich aber auch herausgestellt, dass wir unser Projekt auch im Rahmen bereits ausgereifter, existierender Technologien realisieren können. Bezüglich der von uns geplanten ortsbasierten Dienste ist eine Ortung mit UMTS zwar genauer möglich, aber die Nutzung der bestehenden Möglichkeiten unter GSM reichen für unsere Anforderungen zunächst aus. Außerdem ist die Schnittstelle zwischen GSM und UMTS so geplant, dass unsere Anwendungen später ohne größere Probleme auf UMTS portiert werden können.
Im weiteren Projektverlauf hat sich herausgestellt, dass die Zusammenarbeit mit einem großen Wirtschaftsunternehmen äußerst zeitintensiv ist. Den Großteil unserer Anwendungen konnten wir jedoch zunächst ohne weitere Hilfestellungen fertig stellen. T-Mobile stellte uns dann später eine API für die ortsbasierten Dienste zur Verfügung. Daraufhin konnten wir auch die Anbindung an diese API größtenteils implementieren. Als es darum ging, unser System im Live-Betrieb zu testen, tauchten jedoch einige Probleme auf. T-Mobile war großzügigerweise bereit, uns den Zugang auf ihren Server und ihre (ortsbasierten) Daten zu gestatten. Wir waren jedoch als studentische Projektgruppe bedauerlicherweise nicht in der Lage, alle rechtlichen Anforderungen (zum Beispiel im Bereich des Datenschutzes) zu erfüllen, so dass wir dieses Angebot leider nicht wahrnehmen konnten.
Damit existiert jedoch die Möglichkeit, dieses Projekt weiterzuführen. Mit mehr Zeit können vielleicht alle nötigen Voraussetzungen für eine weitere Zusammenarbeit mit T-Mobile geschaffen werden. Dann kann die bereits implementierte Anbindung an die LBS-API eingebunden und getestet werden, und am Ende könnte ein erfolgreicher Live-Betrieb unseres Projekts stehen.

Kapitel 5

Qualitätssicherung

Wie bereits beschrieben, wurde zu Beginn der Projektgruppenarbeit zusammen mit der Projektleitung und mit Zustimmung aller Teilnehmer ein Zeitplan entwickelt, um einen Überblick über die noch zu erledigenden Aufgaben zu erhalten und frühzeitig einen Verzug erkennen zu können. Deswegen konnten sehr gut frühzeitig Maßnahmen ergriffen werden, um die Qualität der Ergebnisse sicherzustellen.
Im Folgenden werden die Maßnahmen und unsere Erfahrungen damit beschrieben. Im Allgemeinen war diese Vorgehensweise sehr erfolgreich, und gestellte Deadlines wurden überwiegend eingehalten. Hin und wieder wurde bei sehr umfangreichen oder dringenden Aufgaben eine Deadline unwesentlich überschritten.

5.1 Schulungen

Um den Wissensstand aller PG-Teilnehmer auf ein gleiches Level zu bringen, wurden Schulungen in Form von Vorträgen durchgeführt. Einzelne Teilnehmer, die sich tiefer mit einem Thema beschäftigt haben, veröffentlichten ihre Erkenntnisse durch Präsentationen in den PG-Treffen.
Vorträge wurden gehalten, um Basiswissen in verschiedenen Bereichen zu vermitteln, wie die Architektur und Funktionsweise von EJB zusammen mit einem Applikationsserver sowie die Benutzung von Programmen wie WinCVS funktionieren.
Im Folgenden sind die Schulungen aufgelistet, die während der Anfangsphase der Projektgruppenarbeit durchgeführt wurden.

- Java 2 MicroEdition [Sun03c] - Schulung
- Enterprise Java Beans [Sun03b] - Schulung
- Application Server - Schulung
- CVS und WinCVS [Cvs] - Schulung
- Latex und Lyx - Schulung

5.2 Versionierung

Um die Versionskontrolle von Dokumenten und Source Code durchzuführen, wird das „Concurrent Versions System“ eingesetzt und zur Bedienung WinCVS auf den PG-

Pool Rechnern installiert. Dadurch ist eine Rückverfolgung der Änderungen zeitlich und nach Personen genau nachvollziehbar.

Am Anfang wurden einige Dokumente in MS Word erstellt, die im CVS nur binär eingecheckt werden können. Dadurch waren leider die Hauptfunktionen, für die man ein CVS einsetzt, nicht verwendbar. Da wären zum Beispiel die Funktionen wie „diff“, um Unterschiede zwischen versionierten Dokumenten anzuzeigen oder auch unterschiedliche Dokumente automatisch zusammenzufügen (merge). So könnten höchstens Binärdokumente im CVS-Repository abgelegt und wiedergefunden werden. Hierfür allein benötigt man jedoch kein CVS.

Das Problem in der PG-Startphase, bis das CVS installiert, funktionsfähig und benutzbar war, wurde vorübergehend gelöst durch eine Revision History zu Beginn jedes binären Dokuments. Hier musste eine Versionsnummer eingetragen werden, das Datum der Änderung, die Namenskennung und eine Beschreibung der durchgeführten Änderungen. Doch sind wir schnell dazu übergegangen, nur Dokumente im Klartext zu erstellen und im CVS einzuchecken.

Hauptsächlich wurde das CVS für die Versionierung des Sourcecodes verwendet. Es wurde auf den lokalen Systemen programmiert und getestet, sodass nur kompilierbare Versionen eingecheckt wurden. Bei dem Erreichen von bestimmten Meilensteinen wie den geplanten Inkrementstufen wurden TAGS gesetzt, so dass der entsprechende Prototyp-Stand jederzeit schnell wiedergewonnen werden konnte.

Am Anfang jeder Datei wurde folgender Kommentar-Kopf eingefügt. Der TAG Log wird beim Auschecken durch die gesamte History des Dokuments ersetzt inklusive Revisionsnummer, Datum des letzen Eincheckens und Namen des Benutzers, sowie dessen Kommentar.

Beispiel einer History:

```
/*
 * Datei: Dateiname.java
 * Letzter Bearbeiter: \$Author: LarsH $
 *
 * Projektgruppe 409, Universität Dortmund
 * mCube - Mobile Community for UMTS Based Entertainment
 *
 * Revision 1.5  2003/03/22 16:43:03  mariokn
 * Zitate hinzugefügt
 *
 * Revision 1.4  2003/03/16 14:52:28  MikaK
 * Neue Rechtschreibung
 *
 * Revision 1.3  2003/03/05 16:49:24  SebastianR
 * Beispielcode in verbatim-Umgebung verpackt
 *
 * Revision 1.2  2003/03/03 08:43:07  mariokn
 * kleine Verbesserungen
 *
 * Revision 1.1  2003/03/02 19:17:29  mariokn
 * initial
 *
 */
```

5.3 JavaDoc

Von dem Source Code soll mit JavaDoc [Sun] eine ständig aktuelle, dokumentierte API generiert werden können. Dazu müssen im Code bestimmte TAGS eingefügt werden. Sie beschreiben die Funktionen der Klassen und Methoden, zeigen den Autor und die aktuelle Version. Wir halten uns an die von Sun definierten TAGS für JavaDoc 1.3.
Für den ersten Prototypen wurde dieses Vorgehen allerdings vernachlässigt, da im Vordergrund das Erkennen und Beseitigen von späteren Problemen stand. Außerdem war der Code noch überschaubar.
Bereits für das erste Inkrement wurde begonnen, alle bis dahin bestehenden Methoden mit JavaDoc-Kommentaren zu versehen. Diese wurden während der Entwicklungszeit phasenweise immer wieder aktualisiert und vervollständigt.

5.4 Code Convention

Für ein einheitliches Erscheinungsbild des erzeugten Source Codes wird die Code Convention von Sun verwendet. Unsere Zusammenfassung der Convention wird durch ein 20 Seiten starkes Dokument beschrieben. Das Dokument beschreibt Konventionen bezüglich der zuvor beschriebenen CVS Schlüsselworte, Deklarationen oder Java-Befehle wie FOR-Schleifen und IF-ELSE-Anweisungen. Ebenso gab es Regeln über das Einrücken von Anweisungen, Leerzeilen und dem Programmierstil. Diese eher kleinlichen Regelungen wurden nicht unbedingt eingehalten, was damit zu erklären ist, dass alle PG-Teilnehmer schon vor dem Projekt viel programmiert haben und sich jeder seinen eigenen Stil angewöhnt hat. Wobei trotzdem zu sagen ist, dass gerade durch die große Programmiererfahrung der Teilnehmer sehr gut lesbarer und übersichtlicher Code produziert wurde. Deshalb wurde es nicht notwendig, die Einhaltung des vorgegebenen Programmierstils zu erzwingen.

5.5 Assessment Teams

Für die Prüfung der Qualität von Dokumenten und Berichten wurden Assessment Teams gebildet, d.h. eine umfangreiche, abgeschlossene Aufgabe (z.B. Visionsdokument, Zwischenbericht, Protokolle...) wurden von Teams (2-3 Personen), die mit der Erstellung der Arbeit nichts zu tun hatten, auf Qualität und vorher festgelegte Ziele überprüft (z.B. Inhalt, formaler Aufbau inkl. Rechtschreibung).

5.6 Tests

Auf Ebene des Java-Source-Codes wurde die Funktionstüchtigkeit der einzelnen Komponenten mit Hilfe des Test-Frameworks JUnit [Obj] überprüft. Damit wurde zum einen das Erstellen der Tests vereinfacht, und zum anderen konnten die Tests automatisiert durch unseren Build-Prozess ausgeführt werden. Für die GUI wurden Use-Cases erstellt und von Hand getestet. Gefundene Fehler wurden an die zuständigen Entwickler gemeldet und von diesen beseitigt. Anschließend wurden die gleichen Tests nochmal durchgeführt.

Kapitel 6

Projektplan

Die folgende Tabelle stellt die wesentlichen Punkte unseres Projektplans dar. Dazu gehören die jeweilige Aufgabe, deren Dauer, Beginn und Fertigstellung sowie die PG-Mitglieder, die jeweils für die Erledigung der Aufgabe verantwortlich waren.

- Projektstart: Montag, 15.04.02
- Projektende: Donnerstag, 28.03.03

Anmerkung: *VG* steht für Vorgänger, **fettgeschriebene Tasks** haben folgende *kursive Tasks* als Unterpunkte, mit M gekennzeichnete Tasks sind Milestones.

Task ID (*VG*)	Task Name	Dauer in Tagen	Start	Ende	Verantwortlich
1	Software im PG-Pool installieren und warten	220	15.04.02	14.02.03	NilsG; LarsH
2	Namens- und Logofindung	1	15.04.02	15.04.02	alle
3	Domainreservierung	1	23.04.02	24.04.02	NilsG
4	Klärung der technischen Möglichkeiten	13,5	15.04.02	02.05.02	YoussefK; MikaK; LarsL
5	Entscheidung über Entwicklungsumgebung und sonst. Tools treffen	8	07.05.02	16.05.02	alle
6	Mapinfo-Schulung durchführen	1	25.04.02	25.04.02	LarsL
7	LaTeX-Schulung durchführen	1	30.04.02	30.04.02	MiriamL; RobinN
8	CVS / WinCVS-Schulung durchführen	1	30.04.02	30.04.02	NilsG
9	J2ME-Schulung durchführen	1	21.05.02	21.05.02	YoussefK
10	EJB-Schulung durchführen	1	16.05.02	16.05.02	ChristianL; DanielP

11 (*24*)	Applikationsserver-Schulung durchführen	1	24.06.02	24.06.02	ChristianL; DanielP
12	Projektplan erstellen	13,5	15.04.02	02.05.02	MiriamL
13	Projektplan warten	220	15.04.02	28.03.03	MiriamL
14	Visionsdokument erstellen	16,5	15.04.02	07.05.02	AnnaF; MarioK; LarsL; RobinN
15	Webseite erstellen und aktualisieren	418	30.04.02	28.03.03	LarsL
16	Präsentation Spiel	9	23.05.02	04.06.02	DanielP
17	Präsentation Dating	9	23.05.02	04.06.02	MarioK
M18	Phase 1 (Vorbereitungen) abgeschlossen	1	21.05.02	21.05.02	
19	Anforderungsdokument erstellen	19	18.04.02	14.05.02	TobiasH; SebastianR; MarioK; DanielP
20	Teilprobleme in die Komponenten aufteilen	10,5	14.05.02	28.05.02	SebastianR
21	Klassendiagramm erstellen	10,5	14.05.02	28.05.02	ChristianL; TobiasF; TobiasH; MarioK; DanielP; SebastianR
22	Architektur entwickeln	11	14.05.02	28.05.02	TobiasF; SebastianR; DanielP; ChristianL
M23	Analysephase abgeschlossen	1	28.05.02	28.05.02	
24 (*22*)	Entscheidung über ApplikationsServer	2	29.05.02	30.05.02	alle
25	Detailliertes Diagramm mit Methoden erstellen	15,5	30.05.02	20.06.02	Architekten und Evangelisten
26	Komplettes Architekturmodell entwickeln	16	30.05.02	20.06.02	TobiasF; SebastianR; ChristianL; DanielP
27	GUI-Prototypen entwickeln	24	02.05.02	04.06.02	TobiasH; YoussefK; MarioK
28	Präsentation für Vodafone erstellen	10	07.06.02	20.06.02	AnnaF; MiriamL; SebastianR
29	Richtlinien für die Entwicklung von Komponenten aufstellen	11	05.07.02	19.07.02	MikaK; TobiasF
30	**Zwischenbericht erstellen**	8	02.07.02	18.07.02	RobinN; Miriam Lindhorst
31	*Vorwort, Zielsetzung, Projektplan, Architektur, Anforderungen*	8	02.07.02	02.07.02	RobinN
32	*Vorwort Architektur, Seminarphase*	8	02.07.02	11.07.02	SebastianR

33	*Prozessmodell, Qualitätssicherung*	8	02.07.02	11.07.02	MiriamL
34	*GUI Prototyp .game*	8	02.07.02	11.07.02	YoussefK
35	*GUI Prototyp .community*	8	02.07.02	11.07.02	TobiasH
36	*GUI Prototyp .dating*	8	02.07.02	11.07.02	MarioK
37	*Entwicklungsumgebung*	8	02.07.02	11.07.02	MikaK
38	*Durchstichprototyp Komponentenbericht*	8	02.07.02	11.07.02	alle
M39	Analyse&Design-Phase abgeschlossen	1	18.07.02	18.07.02	
40 (*24*)	**Prototypen entwickeln**	22	13.06.02	12.07.02	alle
41	*Kommunikation und Chat*	22	13.06.02	12.07.02	TobiasH; DanielP
42	*Dating*	22	13.06.02	12.07.02	YoussefK; LarsH
43	*Community, Logging und Timer*	22	13.06.02	12.07.02	AnnaF; MarioK; SebastianR
44	*Game*	22	13.06.02	12.07.02	TobiasF; RobinN
45	*UserMatching, FriendsAround, Routing und Locationing*	22	13.06.02	12.07.02	ChristianL; MikaK; MiriamL; LarsL; NilsG
46	**1. Inkrement**	76	22.07.02	4.11.02	Einteilung wie beim Prototypen
47	*Usermanagement und Groupmanagement verfeinern*	76	22.07.02	4.11.02	MikaK; ChristianL
48	*Matching-Algorithmus ausarbeiten*	76	22.07.02	4.11.02	LarsL; MiriamL
49	*JavaDoc für jede Komponente erstellen*	76	22.07.02	4.11.02	alle
50	*Funktionen, insbesondere der core-Komponenten, fertigstellen*	76	22.07.02	4.11.02	core-Leute
51	*JUnit-Tests für core-Komponenten*	76	22.07.02	4.11.02	core-Leute
52	Nachbesserung 1. Inkrement	29	4.11.02	12.12.02	alle
M53	Präsentation 1. Inkrement	1	12.12.02	12.12.02	alle
54	XSLT-Lösung für das Problem unterschiedlicher Endgeräte erarbeiten	16	16.12.02	6.01.03	NilsG; MiriamL
55	2. Inkrement	41	25.11.02	20.01.03	alle

56	JUnit-Tests für core-Komponenten fertigstellen	21	20.01.03	17.02.03	core-Leute
57	GUI Browser/Handy	33	2.01.03	17.02.03	Anwendungsleute
58	Testen	16	3.02.03	24.02.03	alle
M59	Implementierung abgeschlossen	1	28.02.03	28.02.03	alle
60	**Abschlussbericht, 1. Version**	16	10.02.03	3.03.03	alle
61	*Installationshandbuch*	16	10.02.03	3.03.03	NilsG; DanielP
62	*Vorwort/Motivation und Zusammenfassung/Ausblick*	16	10.02.03	3.03.03	RobinN
63	*Benutzerhandbuch - Administration*	16	10.02.03	3.03.03	NilsG; MikaK
64	*Benutzerhandbuch - Bedienung*	16	10.02.03	3.03.03	Anwendungsleute
65	*Prozessleitfaden und Projektplan*	16	10.02.03	3.03.03	MiriamL
66	*Qualitätsmanagement*	16	10.02.03	3.03.03	MarioK
67	*Entwurf, Systemarchitektur (insbes. auch XSLT)*	16	10.02.03	3.03.03	SebastianR
68	*Details der Implementierung*	16	10.02.03	3.03.03	alle
69	*Testbeschreibungen*	16	10.02.03	3.03.03	alle
70	*Entwicklungsumgebung*	16	10.02.03	3.03.03	NilsG; MikaK
71	Abschlussbericht, Endversion	15	10.03.03	28.03.03	DanielP
72	Abschlusspräsentation vorbereiten	15	10.03.03	28.03.03	alle
73	Ergebnis präsentieren	1	28.03.03	28.03.03	alle
M74	Projektgruppe abgeschlossen	1	28.03.03	28.03.03	

Kapitel 7

Anforderungsanalyse

Die Entwicklung einer umfassenden Anforderungsanalyse war ein wesentlicher Schritt unserer Vorgehensweise. Ziel war es, alle Grundfunktionalitäten zu entwickeln, ohne etwas zu vergessen. Dafür haben wir uns nach allgemeinem Brainstorming in kleine Gruppen aufgeteilt, die dann jeweils die einzelnen Bereiche aufgearbeitet haben.

7.1 Namensschema

Die Anforderungen werden zur einfacheren Wiedererkennung systematisch bezeichnet. Jede Bezeichnung beginnt mit „/“ gefolgt von einem Kürzel des Anforderungsbereiches, also beispielsweise /COM für den Bereich Community. Dann folgen, jeweils mit „/“ getrennt, ein Symbol für die Art der Anforderung (F = funktional, N = nicht funktional, S = Schnittstelle) zusammen mit einer laufenden Nummer sowie eine laufenden Nummer für detaillierte Anforderungen.

Beispiel: „/COM/F3/3“ steht für die dritte Anforderung aus dem dritten großen Paket im Bereich Community.

7.2 Anforderungen Community-Basisdienste

Die Community mCube stellt Basisdienste zur Verfügung, die jederzeit von allen Nutzern beansprucht werden können, aber auch als Komponenten in größeren Anwendungen eingesetzt werden können.

7.2.1 Funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/COM/F1	Nutzer haben die Möglichkeit mit anderen angemeldeten Nutzern zu chatten.	MUSS	Chatten als reine Kommunikation oder zur Verabredung weiterer Aktionen ist ein sehr beliebtes Element in Communitys und wird in allen Bereichen benötigt.

/COM/F1/1	Es gibt die Möglichkeit, sich mit anderen angemeldeten Nutzern direkt zu unterhalten.	MUSS	Zur Gewährleistung der Privatsphäre sind Einzelgespräche notwendig.
/COM/F1/2	Es gibt verschiedene themenbezogene Chats, an denen zur gleichen Zeit viele Nutzer teilnehmen können.	KANN	In themenbezogenen Chats ist es einfach, Menschen mit gleichen Interessen kennen zu lernen.
/COM/F1/3	Der Nutzer kann Listen anderer Nutzer verwalten, von denen Informationen zu Anmeldestatus und Aufenthaltsort angezeigt werden.	MUSS	
/COM/F1/4	Ein Nutzer kann einem anderen eine Nachricht schicken, die asynchron gelesen und beantwortet werden kann.	MUSS	Durch asynchronen Nachrichtenaustausch können auch nicht angemeldete Nutzer teilnehmen.
/COM/F1/5	Die geschickten Nachrichten können Bilder enthalten.	KANN	Wichtig für das Dating.
/COM/F1/6	Die geschickten Nachrichten können Multimediaelemente enthalten.	KANN	Multimedia-Nachrichten wird ein hohes Potenzial zugeschrieben. Daher sollten wir versuchen, Unterstützung dafür anzubieten.
/COM/F2	Nutzer haben die Möglichkeit die Positionen von Freunden und Objekten zu ermitteln.	MUSS	FriendsAround ist ein interessanter Location Based Service und ein echter Mehrwert im Vergleich zu nicht mobilen Communitys.
/COM/F2/1	Nutzer haben die Möglichkeit die absolute Position und Entfernung eines anderen Nutzers oder Objektes zu ermitteln.	MUSS	

/COM/F2/2	Ein Nutzer wird alarmiert, wenn sich ein anderer Nutzer aus einer definierten Gruppe innerhalb einer gewissen Entfernung aufhält.	MUSS	So kann man sich mit Freunden treffen, wenn diese zufällig in der Nähe sind.
/COM/F2/3	Die Aufenthaltsorte von anderen Nutzern oder Objekten können live auf einer Karte verfolgt werden. Der eigene Aufenthaltsort befindet sich dabei in der Kartenmitte. Der Kartenausschnitt kann verschoben und skaliert werden.	KANN	
/COM/F2/4	Zu angezeigten Nutzern oder Objekten können auf der Karte Zusatzinformationen angezeigt werden.	KANN	Insbesondere bei touristischen Objekten sind Zusatzinformationen nützlich.
/COM/F2/5	Objekte können von Nutzern anhand von Positionsdaten definiert werden.	KANN	Nutzer können so eigene Lesezeichen auf einer Karte anlegen.
/COM/F2/6	Objekte können von neuen Diensten definiert und in Benutzungsabläufe integriert werden.	MUSS	
/COM/F2/7	Ein Nutzer kann Objekte und Personen suchen, die eine gewisse Entfernung um einen von ihm definierten Ort besitzen.	KANN	Nicht immer ist die aktuelle Position interessant, sondern die, die man in Kürze erreicht.
/COM/F3	Der Nutzer kann andere Nutzer suchen, die gewisse Eigenschaften im Profil vorweisen. (Näheres im Abschnitt Administration)	MUSS	Grundlage beispielsweise für Dating Anwendungen.

/COM/F3/1	Jede Eigenschaft im Profil eines Nutzers kann als Suchkriterium dienen. Zudem können Bereiche und boolesche Funktionen zur Kombination angegeben werden. Eine Eigenschaft ist dabei ausdrücklich auch die Position.	MUSS	
/COM/F3/2	Suchen können abgespeichert werden, damit sie später wiederholt ausgeführt werden können.	KANN	
/COM/F3/3	Suchen können im Hintergrund aktiviert werden. Der Nutzer wird benachrichtigt, wenn die Bedingungen eingetreten sind.	KANN	So kann einfach verfolgt werden, ob neu angemeldete Personen den Kriterien entsprechen.
/COM/F4	Nutzer können anderen Nachrichten hinterlassen, die aktiviert werden, wenn die Adressaten an bestimmten, vom Absender definierten Orten sind.	MUSS	Interessante Idee, neue Form von Kommunikation.
/COM/F4/1	Die hinterlassenen Nachrichten lassen sich an weitere Bedingungen wie Zeit, alte Position(en) oder die gleichzeitige Anwesenheit anderer knüpfen.	KANN	Bietet sehr viel mehr Möglichkeiten in Anwendungen.
/COM/F5	Es existiert eine Währung namens Cubies, mithilfe derer man besondere Dienste in Anspruch nehmen kann.	KANN	Durch die Einführung einer Währung können kostenpflichtige Dienste erbracht oder Bonussysteme eingeführt werden.

/COM/F5/1	Transaktionen und Kontostand können jederzeit auf einem Konto eingesehen werden.	KANN	
/COM/F5/2	Cubies können an andere Nutzer verschenkt werden.	KANN	
/COM/F6	Der Nutzer muss zu jeder Zeit wissen und kontrollieren können, welche seiner Daten anderen Nutzern und dem System zugänglich sind. Zu diesen Daten zählen: Aufenthaltsort, Anmeldestatus, Realname, Aliasname, Persönliche Daten, Kontaktinformationen.	MUSS	Jeder Mensch hat das Recht auf informationelle Selbstbestimmung. Zum einen wird diese gesetzlich geschützt, zum anderen hängt die Akzeptanz der Dienste vom Gefühl der Wahrung einer gewissen Privatsphäre ab.
/COM/F6/1	Der Nutzer kann einsehen, wann seine Positionsdaten abgefragt wurden.	KANN	Die Positionsdaten sind sehr sensible Daten, über die der Nutzer volle Kontrolle haben möchte. Durch diese Kontrolle ist es nicht möglich, jemanden unbeobachtet auszuspionieren.
/COM/F6/2	Der Nutzer kann einstellen, wie häufig seine Positionsdaten abgefragt werden.	MUSS	Durch allzu häufige Abfrage der Positionsdaten wird der Stromverbrauch extrem erhöht.

/COM/F7	Routing: Es stehen verschiedene Varianten zur Führung zur Verfügung: • Ein Nutzer wird an einen bestimmten Ort geführt • Zwei Nutzer werden gemeinsam an einen bestimmten Ort geführt • Zwei Nutzer werden zusammengeführt Der genaue Ablauf dieser Führung muss noch festgelegt werden.	MUSS	Das Routing eines Nutzers ist ein zentraler Bestandteil für Location Based Services, der Voraussetzung für fast alle komplexeren Anwendungen sein dürfte.
/COM/F7/1	Die Route zu einem Ort wird straßengenau berechnet und auf einer Karte und textuell angezeigt.	KANN	Ohne fremde Komponenten wohl kaum zu realisieren.
/COM/F8	Die einzelnen Anwendungen sind personalisierbar, der Nutzer kann also Einstellungen speichern, die beim nächsten Login übernommen werden.	MUSS	Weniger Eingaben nötig, insbesondere bei Handys wertvoll.
/COM/F8/1	Alle Anwendungen sind lokalisierbar, d.h. insbesondere die Sprache der Texte ist einstellbar.	KANN	

7.2.2 Nicht funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/COM/N1	Die bei den funktionalen Anforderungen identifizierten Komponenten müssen einfach in größere Anwendungen einzubetten sein.	MUSS	Die von uns entwickelten Komponenten sind Kernkomponenten für mobile Anwendungen und sollten daher auch weiterhin genutzt werden können.
/COM/N2	Ein Verbindungsabbruch darf nicht zum Abbruch eines komplexen Anwendungsfalls führen.	KANN	Da Verbindungsabbrüche bei Einführung von UMTS nicht auszuschließen sind, muss besonders darauf geachtet werden, welche Aktionen abgebrochen und welche aufrechterhalten werden sollten
/COM/N3	Die Konten der elektronischen Währung dürfen nicht manipulierbar sein.	KANN	Heikler Punkt, insbesondere wenn Partner Geld damit verdienen wollen.
/COM/N4	Die Darstellung sämtlicher Inhalte muss sowohl auf einem mobilen Endgerät als auch in einem Webbrowser möglich sein.	MUSS	

7.2.3 Schnittstellenanforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/COM/S1	Die einzelnen Dienste müssen Schnittstellen anbieten, um in komplexeren Anwendungen wieder verwendet werden zu können. Diese Schnittstellen lassen sich bei allen Komponenten einheitlich verwenden.	MUSS	Schlüsselanforderung, um eine Plattform zu schaffen in der weitere Anwendungen entstehen können.

/COM/S2	Der Routing-Dienst benötigt Informationen vom Location-Server des UMTS-Netzes, muss also die dort definierte Schnittstelle nutzen.	MUSS	

7.3 Anforderungen Subsystem mCube.game

Die Komponente mCube.game realisiert den Gedanken eines ’Trading-Card’ ähnlichen Spiels aufsetzend auf den in Absatz 3 beschriebenen Community Funktionalitäten.

7.3.1 Voraussetzungen

Vorausgesetzt wird die vollständige Implementierung folgender Funktionalitäten des mCube-Systems:

- Anzeige spielbereiter Teilnehmer unter Angabe derer Entfernung [MUSS]
- Verwaltung von Profilen inkl. .game-relevanter Daten [MUSS]

An die **Darstellung** unter *.game* werden folgenden Voraussetzungen gestellt:

- Existenz eines Hauptbildschirms, der den Ausgangspunkt bei Wechsel in den Spielmodus aus der Community heraus darstellt.
- Existenz eines Spielbildschirms, der alle Funktionalitäten eines laufenden Spiels darstellt.
- Existenz eines Kartenbildschirms, der zur Darstellung aller verwaltungsspezifischen Funktionalitäten bezüglich des Kartensatzes eines Users dient.

7.3.2 Spielregeln

Spielcharaktere besitzen vier **Grundeigenschaften**:

- Konstitution
- Geschicklichkeit
- Stärke
- Intelligenz

Es existieren verschiedene **Grundcharaktere**, die sich in der Verteilung von anfänglich insgesamt 20 Punkten auf oben genannte Grundeigenschaften, sowie in der Anzahl und Art der ihnen zu Verfügung stehenden Aktionen unterscheiden.
Gekämpft wird **rundenbasiert** solange, bis die Anzahl der erlittenen Treffer die Anzahl der dem Charakter zu Verfügung stehenden **Schadenspunkte** übersteigt. Die Schadenspunkte errechnen sich nach der Formel: Schadenspunkte $= 5 \cdot$ Konstitution. Vor

Charakter	Konstitution	Geschicklichkeit	Stärke	Intelligenz
Elf	3	7	4	6
Magier	3	6	3	8
Barbar	7	3	7	3
Paladin	4	5	6	5

Tabelle 7.4: Initiale Punkteverteilung Charaktere

der Ausführung eines Zuges gibt der Spieler an, wie offensiv / defensiv seine Aktion ausgeführt werden soll. Dazu wählt er den **Offensivwert von 0 - 100%** aus. **Ein Offensivwert von 0%** (also die reine Verteidigung) kann dabei als Abwehr durch Geschicklichkeit (Ausweichen) gewertet werden.

Der **Kampfwert** eines Charakters errechnet sich aus den für die ausgewählte Waffe zuständigen Charakterwerten. **Offensiv- und Defensivwert der Waffe** errechnen sich durch Aufschläge (Boni) auf den Kampfwert. Der **Gesamtdefensivwert** berechnet sich zu: $\text{Gesamtdefensivwert} = \text{Rüstungsdefensivwert} + \text{Waffendefensivwert}$.

Beispiel zur Berechnung eines Spielzuges: Spieler A hat einen Waffen-Offensivwert von 12 und spielt zu 80% offensiv.

Spieler B hat einen Gesamt-Defensivwert von 18 und spielt zu 50% defensiv.

80% von 12 sind 9.6

50% von 18 sind 9.0

Differenz: 0.6

Aus der Formel: $\text{Basis} = 50 - 5 * (\text{Offensivwert}(\text{Angreifer}) - \text{Defensivwert}(\text{Verteidiger}))$ errechnet sich der Wert 47.

Spieler A würfelt nun mit **1W100** (Wertebereich: 1-100). Ist das Ergebnis kleiner oder gleich der Basis, so war der Angriff erfolglos. War der Wurf größer als die Basis, errechnet sich der **erzielte Schaden** nach:

$$\text{Schaden}(\text{Waffe}) = \left[\frac{(\text{Wurf} - \text{Basis} - 1)(\text{Maxwert}(\text{Waffe}) - \text{Minwert}(\text{Waffe}) + 1)}{(100 - \text{Basis})}\right] + \text{Minwert}(\text{Waffe})$$

Für eine Basis ≥ 100 zählt eine 100 dennoch als Erfolg, der Minimalschaden anrichtet.
Für das Beispiel ergibt sich aus der Formel folgende Schadenstabellen:

Wurf	Schaden (Keule)
48-53	1
54-58	2
59-63	3
64-69	4
70-74	5
75-79	6
80-85	7
86-90	8
91-95	9
96-100	10

Wurf	Schaden (Schwert)
48-65	4
66-83	5
84-100	6

Wurf	Schaden (Ritzkihutu)
48-61	2
62-74	3
75-87	4
88-100	5

Wurf	Schaden (Stab)
48-65	1
66-83	2
84-100	3

Der **Schaden** wird nun von den **Schadenspunkten** von Spieler B abgezogen und obige Rechnung für die Offensive von Spieler B (mit A in der Defensive) wiederholt.

Erreichen die Schadenspunkte eines Spielers am Ende einer vollständigen Runde 0, so hat er den Kampf verloren. Erreichen die Schadenspunkte beider Spieler 0, endet das Spiel unentschieden.

Der Charakter eines Spieles stirbt nicht, die Schadenspunkte eines Spielers werden bei jedem neuen Kampf voll aufgefüllt.

7.3.2.1 Gegenstandskarten:

Schwert	abhängig von:	Stärke, Geschicklichkeit
	Offensivbonus:	2
	Defensivbonus:	2
	Minwert:	4
	Maxwert:	6
	Einschränkung:	Stärke ≥ 6
Stab	abhängig von:	Stärke, Geschicklichkeit
	Offensivbonus:	0
	Defensivbonus:	5
	Minwert:	1
	Maxwert:	3
	Einschränkung:	Nur Magier
Keule	abhängig von:	2×Stärke
	Offensivbonus:	5
	Defensivbonus:	0
	Minwert:	1
	Maxwert:	10
	Einschränkung:	Stärke ≥ 7
Ritzkihutu	abhängig von:	2×Geschicklichkeit
	Offensivbonus:	3
	Defensivbonus:	3
	Minwert:	2
	Maxwert:	5
	Einschränkung:	Nur Elf

Lederrüstung	Defensivbonus: Besonderheit:	3 Geschicklichkeit-1
Eisenrüstung	Defensivbonus: Besonderheit:	5 Geschicklichkeit-2
Amulett der Stärke	Besonderheit:	Stärke+1
Amulett der Konstitution	Besonderheit:	Konstitution+1
Amulett der Intelligenz	Besonderheit:	Intelligenz+1
Amulett der Geschick-lichkeit	Besonderheit:	Geschicklichkeit+1

Waffenkarten werden für jede Runde neu ausgewählt und dürfen beliebig oft benutzt werden. Amulette, Rüstungen, Ringe etc. gehören zum Basisinventar und müssen nicht für jeden Kampf separat ausgewählt werden. Möglich ist hierbei eine Maximalanzahl. So können zeitgleich nur eine Rüstung, ein Amulett und zwei Ringe getragen werden.

7.3.2.2 Ereigniskarten

Heiltrank	Wirkung:	5-10 Schadenspunkte zurück
Ausweichen	Wirkung:	kein Schaden, +5 Offensivbonus in nächster Runde
Ausrutschen	Wirkung:	Gegner fällt hin, 3 Schadenspunkte Gegner, -5 Defensivbonus Gegner in der nächsten Runde

Pro Kampfrunde kann eine Ereigniskarte ausgespielt werden. Eine Ereigniskarte kann pro Kampf nur einmal eingesetzt werden.

7.3.2.3 Magiekarten

Feuerball	abhängig von: Offensivbonus: Defensivbonus: Minwert: Maxwert: Einschränkung:	2×Intelligenz 3 0 1 10 Magier
Todeshauch	abhängig von: Offensivbonus: Defensivbonus: Minwert: Maxwert: Einschränkung:	2×Intelligenz 2 0 5 5 Elf, Magier
Mauer	Defensivbonus: Einschränkung:	10 Elf, Magier, Paladin

Magiekarten werden wie Waffenkarten eingesetzt, sind allerdings auf die magiefähigen Charaktere beschränkt.

7.3.2.4 Erfahrungspunkte

Die Spieler starten mit **0 Erfahrungspunkten auf Level 1.**
Pro Kampf erhält jeder Spieler **5 Erfahrungspunkte.**
Der Sieger erhält zusätzlich $5 + 10 \cdot \text{Leveldifferenz}$ Erfahrungspunkte, wenn sein eigenes Level niedriger ist, bei gleichem oder höherem Level erhält der Sieger 5 Punkte zusätzlich.
Der Level eines Charakters berechnet sich aus den Erfahrungspunkten, die er gesammelt hat. Wir definieren einen Levelaufstieg rekursiv (folgende Formel ist für den Spielablauf sinnvoll und kann so hingenommen werden):
$\text{Levelaufstieg}(1) = 0$
$\text{Levelaufstieg}(k) = \text{Levelaufstieg}(k-1) + \lfloor 1{,}5^{k-2} \cdot 100 \rfloor$ für $k \geq 2$. Daraus ergibt sich:

Level 1:	0-99
Level 2:	100-249
Level 3:	250-474
Level 4:	475-811
Level 5:	812-1317
...	...

Pro Levelaufstieg darf der Spieler eine Charaktereigenschaft um 1 erhöhen.

7.3.3 Funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/GAM/F1/1	Nutzer haben die Möglichkeit in den Hauptbildschirm des Spieles zu wechseln.	MUSS	Anforderungen F2 beziehen sich auf Aktionen aus dem Hauptbildschirm heraus.
/GAM/F2/1	Nutzer können in den Kartenbildschirm wechseln.	MUSS	
/GAM/F2/2	Nutzer können das Spiel beenden.	MUSS	Wechsel in den Community-bildschirm.
/GAM/F2/3	Nutzer können die Spielart auswählen.	KANN	Anfänglich wahrscheinlich Beschränkung auf einen Spielmodus sinnvoll.
/GAM/F2/4	Nutzer können einen anderen Nutzer herausfordern, dabei kann ein kurzer Text eingegeben werden.	MUSS	
/GAM/F2/5	Nutzer können die Herausforderung eines anderen Nutzers annehmen.	MUSS	

/GAM/F2/6	Nutzer können nach Mitspielern suchen.	MUSS	Suchparameter könnten sein: Spielstufe, Entfernung.
/GAM/F2/7	Nutzer können das Spiel beginnen.	MUSS	Wenn alle notwendigen Einstellungen / Mitspieler vorhanden.
/GAM/F3/1	Nutzer haben die Möglichkeit sich Orte anzeigen zu lassen, an denen sie sich Karten verdienen können.	KANN	
/GAM/F3/2	Nutzer können durch ihren Kartensatz blättern, eine Karte enthält dabei die in den Spielregeln beschriebenen Informationen.	MUSS	
/GAM/F3/3	Nutzer können eine oder mehrere Karten auswählen und diese mit einem ausgewählten Nutzer tauschen.	MUSS	Auswahl zwischen IR und 'online' Tauschen? TBD
/GAM/F3/4	Nutzer können tauschwillige Mitspieler suchen.	MUSS	siehe /GAM/F3/3
/GAM/F3/5	Nutzer können in der Community ein Tauschangebot hinterlassen.	KANN	Ist wahrscheinlich Grundfunktionalität der Community.
/GAM/F3/6	Nutzer können den Kartenbildschirm verlassen.	MUSS	Wechsel in Hauptbildschirm.
/GAM/F3/7	Nutzer können direkt in die Community wechseln.	MUSS	Wechsel in Communitybildschirm.
/GAM/F4/1	Nutzer können die Einstellungen für ihren nächsten Zug vornehmen, die Möglichkeiten für die Beeinflussung eines Zuges ergeben sich aus den Spielregeln.	MUSS	Anforderungen F4 beziehen sich auf Aktionen aus dem Spielbildschirm heraus, Abhängig von Spielregeln (siehe 7.3.2).

/GAM/F4/2	Nutzer können den gewählten Zug durchführen.	MUSS	
/GAM/F4/3	Nutzer können das Spiel beenden (Wechsel in den Hauptbildschirm).	MUSS	
/GAM/F4/4	Nutzer können während des gesamten Spieles Nachrichten untereinander austauschen.	KANN	Sollte 'IRC'-ähnlich aussehen.
/GAM/F4/5	Dem Nutzer stehen Hotkeys mit Standardnachrichten zur Verfügung.	KANN	Also 'Beleidigungen' usw.
/GAM/F4/6	Die Hotkeys für Standardnachrichten können vom Benutzer angepasst werden.	KANN	

7.3.4 Nicht funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/GAM/N1	Das System muss die Gültigkeit eines Spielzuges überprüfen können.	MUSS	
/GAM/N2	Das System kann dem Nutzer den gegnerischen Zug anzeigen.	MUSS	
/GAM/N3	Konsequentes Ablehnen von Herausforderungen wird 'bestraft'.	KANN	
/GAM/N4	Die Kartensätze der Spieler sind vom Nutzer nicht manipulierbar.	MUSS	
/GAM/N5	Der Kartensatz kann lokal gespeichert und damit das Spiel über IR auch offline gespielt werden.	KANN	

/GAM/N6	Bei Gewinnen eines Spiels erhält der Nutzer eine, in den Spielregeln festzusetzende, 'Belohnung'	MUSS	
/GAM/N7	Die Tauschfunktionalität darf nicht manipulierbar sein.	MUSS	
/GAM/N8	Im Spielbildschirm existiert eine History der zuletzt durchgeführten Züge.	KANN	

7.3.5 Schnittstellenanforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/GAM/S1	Die Schnittstellen müssen kompatibel zu denen der Community-Plattform sein	MUSS	Komponentengedanke der Community

7.3.6 Auftretende Klassen (vorläufig)

1.	Nutzer	Der Benutzer des Systems
2.	Kartensatz	Sammlung der Spielkarten eines Nutzers
3.	Karte	Spielkarte
4.	Chattext	Kurznachricht
5.	Hotkey	Belegt mit Kurznachrichten
6.	Hauptbildschirm	Hauptschirm des Spiels
7.	Spielbildschirm	Schirm bei laufendem Spiel
8.	Kartenbildschirm	Anzeige des Kartensatzes
9.	Layout	Layout eines Schirmes
10.	Ort	Ort, an dem Karten verdient werden können
11.	Spiel	Status des laufenden Spieles
12.	Spielart	Art des Spiels (Spielregeln)
13.	Zugeinstellung	Einstellung für den nächsten Zug
14.	Zug	Besteht aus allen gemachten Zugeinstellungen

7.4 Anforderungen Subsystem mCube.dating

mCube.dating stellt einen Service zur Verfügung mit dem Ziel jemanden (un-)bekannten zu treffen. Der Service führt den User von der ersten Kontaktaufnahme bis hin zu einem vom System ausgewählten Treffpunkt. Es werden die Features der darunter liegenden Community genutzt.

Nicht aufgeführte Anforderungen, die in der Community beschrieben sind, können übernommen werden.

Zur Begriffsdefinition: Es wird zwischen Partner und Freund unterschieden, da der Dienst mit beiden Suchabsichten umgehen, und natürlich differenzieren können sollte.

7.4.1 Funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/DAT/F1	Der User hat die Möglichkeit Profile anzulegen.	MUSS	
/DAT/F1/1	Eigenes Dating-Profil anlegen/ändern	MUSS	Das Dating-Profil sollte nicht alleine stehen, sondern ergänzend zu dem bereits vorhandenen Community-Profil sein.
/DAT/F1/2	Dating-Profil des zu suchenden Date-Partners anlegen/ändern. (Falls man jemanden sucht, der einem selbst nicht ähnlich ist.)	MUSS	Die Profile sollten aus wenigen Bereichen bestehen, wie äußere Erscheinung, Charakter, Interessen.
/DAT/F1/3	Die Items in den Profilen sollten vorgegeben und aus einer Combobox gewählt werden können. So könnte das Date-Partner Profil nur aus Angaben größer, kleiner, jünger, älter bestehen.	KANN	Das erleichtert den Abgleich von Profilen und die Rankingerstellung.
/DAT/F1/4	In den Profilen kann Freund oder Partner gewählt werden.	MUSS	Um auch Personen suchen zu können, mit denen man keine sexuelle Beziehung anstrebt.
/DAT/F1/5	In das eigene Profil kann ein Foto hochgeladen oder aus einer Menge charakterbeschreibender Comic-Bilder ausgewählt werden.	KANN	
/DAT/F2	Es gibt eine Dating-Liste.	KANN	Hier werden im Dating registrierte User aufgelistet

/DAT/F2/1	Der User kann seinen Date-Partner aus der Liste wählen, anstatt ihn suchen zu lassen.	KANN	So hat der User auch die Freiheit manuell zu wählen. Schwierig könnte die Auflistung inklusive der Eigenschaften sein.
/DAT/F2/2	Man kann seine eigene Buddyliste anlegen, um potenzielle Date-Partner zu sammeln.	KANN	Diese kann angelehnt werden an ICQ oder MSN Messenger.
/DAT/F3	Bereich auswählen, in dem gesucht werden soll.	MUSS	Die Bereichsangabe sollte möglichst nicht in km sondern in Stadt oder Stadtteilen angegeben werden können.
/DAT/F4	Wurde jemand für ein Date ausgewählt, kann er annehmen oder ablehnen.	MUSS	Wurde die Einladung abgelehnt, sollte es für eine bestimmte Zeit nicht möglich sein, den Date-Partner nochmals anzuschreiben. Das verhindert Belästigungen.
/DAT/F5	Es kann mit dem Date-Partner gechattet werden.	MUSS	Es können alle Funktionen vom Community-Chat übernommen werden
/DAT/F6	Date-Partner Suche ausführen.	MUSS	Mit allen zuvor eingestellten Werten.
/DAT/F7	Automatische Lokalisierung ein-/ausschalten	MUSS	Von der Community übernehmen.
/DAT/F8	Der User kann noch nicht registrierte Personen zu mCube.dating einladen.	MUSS	Mit vollendeter Registrierung dieser Person erhält der User eine hohe Anzahl an Cubies auf sein Konto gutgeschrieben.
/DAT/F9	Zu einem Treffen kann ein Treffpunkt gewählt werden.	MUSS	Es werden alternative Treffpunkte in einer Liste dargestellt.
/DAT/F9/1	Als Beweis, dass sich zwei User tatsächlich getroffen haben, muss das Treffen bestätigt werden.	KANN	Auf diese Weise kann auch jedem User Cubies auf deren Konten gutgeschrieben werden.
/DAT/F9/2	Cubies können am Treffpunkt eingelöst werden.	KANN	Problem, ob man das per Infrarot an der Kasse oder dem Barman oder als TAN oder zugesendetem Gutschein.

/DAT/F10	Es können Dating-Gruppen angelegt werden.	KANN	So kann man zwei Gruppen zusammenführen, oder es kann sich jemand einer Gruppe anschließen. Es können nur registrierte User einer Gruppe hinzugefügt werden. Bei einem Treffen erhalten alle Gruppenmitglieder Cubies gutgeschrieben.
/DAT/F11	Der Kontostand kann jederzeit eingesehen werden.	MUSS	
/DAT/F12	Können zwei User näher zusammen nicht geortet werden, erhalten beide diese Nachricht, zusammen mit der Entfernung.	MUSS	In der Stadt können das 10m und auf dem Land mehrere Kilometer sein. Da ist es schon schwieriger den anderen zu finden.

7.4.2 Nicht funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/DAT/N1	Es sollte zu jeder realen Person nur ein Profil geben.	MUSS	Die Registrierung könnte per Post gesendet werden. Auch eine Person mit zwei Endgeräten sollte keine zwei Profile anlegen können.

7.5 Anforderungen Subsystem mCube.admin

mCube.admin soll den kompletten Administrationsservice zur Verfügung stellen. Dieser ist sowohl für einen (neuen) User gedacht, der sich an- bzw. abmelden will, als auch für den Administrator, der Einsicht in die verschiedenen Profile erhalten soll.

7.5.1 Funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/ADM/F1	Der User hat die Möglichkeit ein Profil anzulegen.	MUSS	
/ADM/F1/1	Neues Anmelden	MUSS	Ein User muss sich neu anmelden können und dabei sein Profil erstellen.

/ADM/F1/2	Unterprofile erstellen. Die Profile werden separat gespeichert. Ein Zugriff auf das Basisprofil ist möglich, so dass Redundanzen vermieden werden.	MUSS	Sollte der User Interesse an mCube.Dating oder mCube.game haben, so können hier auch gleich deren Profile bearbeitet werden
/ADM/F1/3	Nutzungs- und Rechtsbelehrung.	MUSS	Der User muss über die Nutzungsbedingungen informiert werden (einschließlich Ortung). Dieser Dialog muss bestätigt werden, soll eine Anmeldung erfolgen.
/ADM/F1/4	Neuanmelden bestätigen.	MUSS	Der User muss seine Neuanmeldung noch einmal bestätigen.
/ADM/F2	Der User hat die Möglichkeit ein Profil zu ändern.	MUSS	
/ADM/F2/1	Bestehendes Profil bearbeiten.	MUSS	Hier soll der User in einem Formular die Möglichkeit bekommen, sein Profil anzupassen.
/ADM/F2/2	Änderungen speichern.	MUSS	Sollte der User Änderungen an dem Formular vorgenommen haben, sollen diese speicherbar sein.
/ADM/F2/3	Änderungen verwerfen.	MUSS	Der User muss eine am Formular vorgenommene Änderung wieder verwerfen können.
/ADM/F2/4	Verlassen.	MUSS	Das Formular muss auch verlassen werden können, ohne das Änderungen vorgenommen wurden.
/ADM/F3	Administratorfunktionen	MUSS	
/ADM/F3/1	Benutzerprofil auswählen	MUSS	Der Admin soll aus einer Liste ein Profil auswählen können, hierzu soll der Name des Users als Schlüssel dienen.

/ADM/F3/2	Benutzerprofil ändern	MUSS	Der Admin kann das Profil eines Users genauso bearbeiten, wie der User selbst, nur das Passwort wird in Klartext dargestellt.
/ADM/F3/3	Benutzerprofil löschen	MUSS	Der Admin kann ein Profil komplett löschen.
/ADM/F3/4	Benutzer informieren	KANN	Der Admin kann dem User eine Nachricht in Form einer Mail zukommen lassen.
/ADM/F4	Der User hat die Möglichkeit, sich vom System abzumelden.	MUSS	
/ADM/F4/1	Vollständiges Abmelden samt löschen aller Profile.	MUSS	Der User hat die Möglichkeit sich vollständig vom System abzumelden. Hierbei werden alle Profildaten gelöscht.
/ADM/F4/2	Deaktivieren einzelner Dienste	KANN	Der User kann sich von einzelnen Diensten wie mCube.Game oder mCube.Dating abmelden und auf Wunsch diese Unterprofile löschen.
/ADM/F5	Kontostand einsehen	MUSS	Der User hat die Möglichkeit seinen Cubie Kontostand einzusehen.

7.5.2 Nicht funktionale Anforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/ADM/N1	Es sollte zu jeder realen Person nur ein Profil geben.	MUSS	Die Registrierung könnte per Post gesendet werden. Auch eine Person mit zwei Endgeräten sollte keine zwei Profile anlegen können.

7.5.3 Schnittstellenanforderungen

Anf. Name	Aufgabenstellung	Typ	Begründung / Anmerkung
/ADM/S1	Sicherheitsaspekte	MUSS	Kein User darf je Zugriff auf die Benutzerdaten anderer User haben.

Kapitel 8

Architektur

Nachdem die Anforderungen an das Projekt formuliert waren, galt es, eine Architektur zu entwerfen, mit deren Hilfe wir diese Anforderungen erfüllen können. Besonders nicht funktionale Anforderungen wie Skalierbarkeit und Integrierbarkeit fremder Anwendungen in unsere Community waren entscheidend für unsere Arbeit.
Auf seiten der Mobilfunkbetreiber werden zumindest zurzeit noch proprietäre Protokolle verwendet. Hier haben wir keinen Einblick in die aktuellen Entwicklungen und können auch zukünftige nicht vorhersehen. Es blieb uns also nur, die Aufrufe an das Netzwerk zu kapseln und den benutzenden Komponenten eine möglichst gute Schnittstelle anzubieten.
Auf der Clientseite, also der Verbindung zum Endgerät, werden die Protokolle und Darstellungssprachen öffentlich diskutiert. Hier ist die Vielfalt der Geräte mit unterschiedlichen Darstellungsgrößen aber auch Darstellungssprachen ein zu bewältigendes Problem. Da alle momentan im Einsatz befindlichen Geräte XML-konforme oder zumindest zu XML sehr ähnliche Sprachen nutzen, ist eine dynamische Erzeugung der Seiten, passend für jedes Endgerät, möglich. Die hierbei von uns verwendete Technik heißt XSLT – Extensible Stylesheet Language for Translations [Cla99]. Dies ist eine sehr mächtige Sprache zur Formatierung von Inhalten, die in XML [BPSM00] beschrieben sind.
Wir haben uns dafür entschieden, komponentenbasiert zu entwickeln. Dadurch können wir logische Einheiten separat entwickeln. Die Integration von Fremdsystemen ist in jedem Entwicklungsstadium problemlos möglich. Die am Markt vorhandenen Komponentenmodelle versprechen zudem Skalierbarkeit, die als Anforderung für einen realen Einsatz sehr wichtig ist.
Unter diesen Voraussetzungen entschieden wir uns für den Einsatz des Produktes Java der Firma Sun in allen Bereichen des Projektes. Java bietet für alle Schichten frei nutzbare Bibliotheken an. Wir nutzen unter anderem die Java 2 Enterprise Edition zur komponentenbasierten serverseitigen Entwicklung, JDBC als Anbindung an die Datenbank und vorhandene API's zur Anbindung an einen LDAP-Server.
Die Technologie Java scheint ausgereift. Es stehen etliche Entwicklungsumgebungen und Applikationsserver zur Verfügung, die wir nutzen können. Zudem vermuten wir, dass sich auch im von uns angestrebten Markt Java als Technologie durchsetzen wird. Auch die Tatsache, dass fast alle PG-Teilnehmer bereits intensive Erfahrungen mit Java gesammelt haben, hat unsere Entscheidung positiv beeinflusst.
Wir erfüllen mit der entwickelten Architektur die an sie gestellten Anforderungen. Sowohl die Schnittstelle zu den Endgeräten als auch die Anbindung an Systeme der Mo-

bilfunkbetreiber ist offen gestaltet, so dass wir problemlos auf verschiedene Techniken reagieren könnten.

8.1 Überblick

8.1.1 Architektur

Als Technik werden die Programmiersprache Java und die von Sun entwickelten Methoden der J2EE (Java 2 Enterprise Edition) [Sha01] eingesetzt. Dies bedeutet den Einsatz eines Applikationsservers, der mit Hilfe von Enterprise Java Beans eine Vielzahl von Anfragen verarbeiten kann. Als Schnittstelle zum Endgerät sollen hauptsächlich die Techniken JSPs und Servlets zum Einsatz kommen, die die inhaltliche Struktur einer Seite erzeugen. Diese inhaltliche Repräsentation wird dann mittels XSLT in eine endgeräteabhängige Darstellung transformiert. Diese Transformation ist für den Anwender weit gehend transparent.
Die Architektur der Plattform mCube gliedert sich in mehrere Schichten. In Abbildung 8.1 ist diese Struktur, die im Folgenden näher beschrieben werden soll, dargestellt:

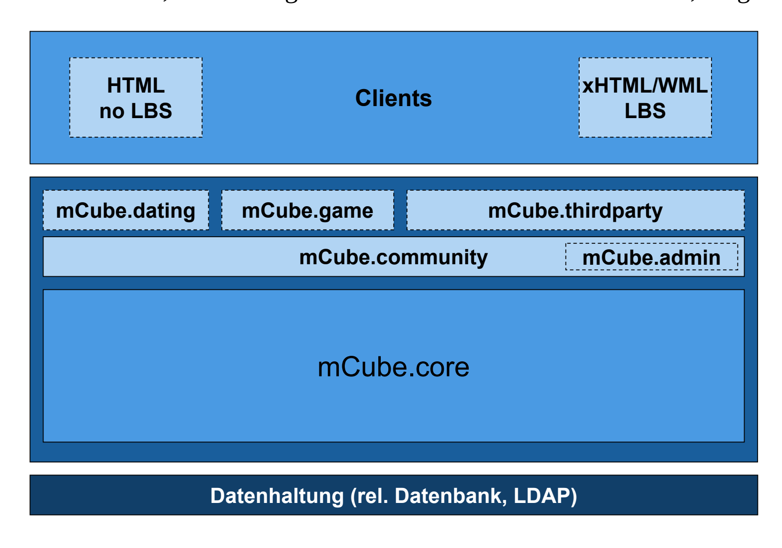


Abbildung 8.1: Struktur der Plattform mCube

8.1.1.1 Die Persistenzschicht

Als Persistenzmechanismus benutzen wir sowohl ein Relationales Datenbankmanagement-System (RDBMS) als auch einen LDAP-Server. Die Nutzerdaten werden in der LDAP-Datenbank gespeichert, weitere Daten wie Nachrichten oder Buddylisten werden in einer relationalen Datenbank gehalten. Der LDAP-Server stellt bereits wichtige Funktionen zur Verwaltung von Nutzerdaten zur Verfügung, eine relationale Datenbank bietet uns darüber hinaus volle Flexibilität und hohe Leistung bei der Verwaltung beliebiger sonstiger Daten.
Die unterschiedlichen Persistenzmechanismen sind nach außen nicht sichtbar. Alle notwendigen Funktionen zur Speicherung und Manipulation von Daten werden von Komponenten der core-Schicht zur Verfügung gestellt.

8.1.1.2 mCube.core

Die unterste logische Schicht bilden einige Komponenten, die gemeinsam den Titel *mCube.core* tragen. Sie bieten alle Funktionen, die die Anwendungen benötigen, um ihre spezifizierten Anforderungen zu erfüllen. Diese Komponenten bilden daher den Kern des Projektes mCube, auf dem alles aufsetzt. Ihre Struktur ist in Abbildung 8.2 dargestellt.

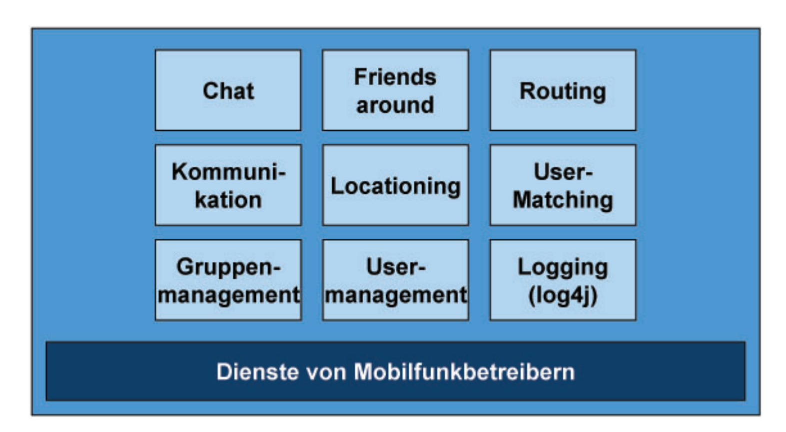


Abbildung 8.2: Struktur mCube.core

Die Anwendungen können die Dienste jeder Komponente aus der core-Schicht direkt nutzen. Wir haben uns gegen den Einsatz einer Fassade entschieden, also einer zusätzlichen Schicht, die oberhalb der core-Komponenten liegt, Anfragen an diese entgegennimmt und diese dann an die einzelnen Komponenten weiterleitet. Eine solche zusätzliche Schicht ist wartungsintensiv, da sie bei jeder Änderung in der Schnittstelle einer Komponente angepasst werden muss. Sie ist unübersichtlich und hebt die logische Trennung der Komponenten auf.
Innerhalb der core-Schicht gibt es keine weitere Struktur, jede Komponente kann auf die Funktionalitäten aller anderen zugreifen.

8.1.1.3 Die Anwendungen

Auf mCube.core setzen verschiedene Anwendungen auf, die die Schnittstelle zum Nutzer bieten. Die Anwendungen können selbst Beans enthalten, die nötige Geschäftslogik implementieren und die Dienste von mCube.core nutzen. Der Kontakt zum Nutzer erfolgt über JSP-Seiten, die durch Servlets gesteuert werden.
Mithilfe dieser Servlets und JSP-Seiten werden XML-Dateien generiert, die im darauf folgenden Schritt per XSLT in beliebige Darstellungssprachen transformiert werden können. Im Rahmen der PG erzeugen wir XHTML- und HTML-Seiten. Dieser Mechanismus wird in Abschnitt 8.1.1.4 erläutert.
Die Anwendung mCube.community setzt direkt auf mCube.core auf und realisiert die Basisdienste der Community wie Anmeldung, Chat und FriendsAround. Ebenso werden die weiteren Anwendungen integriert. So wird ein einheitliches Look & Feel in allen Anwendungen ermöglicht und es Drittanbietern (mcube.thirdparty) erleichtert, an der Plattform zu partizipieren.
Die einzelnen Anwendungen .community, .game und .dating werden in späteren Abschnitten beschrieben.

8.1.1.4 Die Sicht der Clients

Die Clients, die bedient werden sollen, sind sehr verschiedenartig. Zum einen unterscheiden sie sich in der Größe der Darstellungsfläche, zum anderen in ihren Fähigkeiten, GPRS-Dienste nutzen zu können.
Dennoch erfolgt der Zugriff auf die Dienste von mCube in aller Regel über eine einzige Schnittstelle. Alle Clients sprechen ein Servlet an, das zunächst eine geräteunabhängige Darstellung der erzeugten Daten in XML generiert. Diese erstellten XML-Dateien werden dann geräteabhängig in passende Seiten transformiert. Die hierbei verwendete Technologie, XSLT, wird im Bereich Model-View-Controller-Konzept beschrieben.
Die Nutzung der GPRS-Dienste, wie die Positionsermittlung, muss allerdings auf anderen Wegen erfolgen. Hierfür bieten die Mobilfunkbetreiber meist proprietäre API's an, die von den Komponenten der core-Schicht angesprochen werden und deren Funktionalität den Anwendungen dann zur Verfügung gestellt wird.

Das verwendete Model-View-Controller - Konzept Zur Behandlung der Anfragen des Clients benutzen wir ein sogenanntes Model-View-Controller Konzept. Bei diesem Konzept sind die Entgegennahme und Verteilung der Clientanfragen (Controller), die Business-Logik (Model) und die Darstellung (View) voneinander getrennt. [Ses99]
Das von uns verwendete Model-View-Controller Konzept sieht man in Abbildung 8.3. Das zugehörige Klassendiagramm ist in Abbildung 8.4 dargestellt.

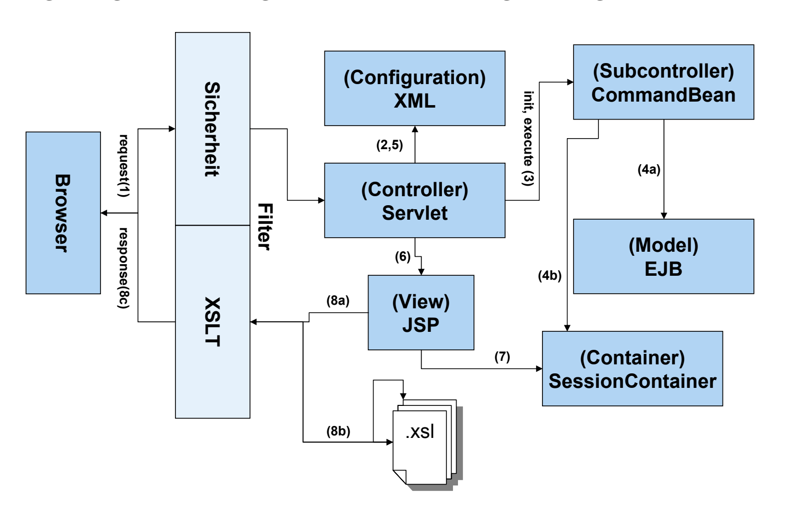


Abbildung 8.3: Unser Model-View-Controller Konzept

Der Browser initialisiert das mCubeServlet und schickt Requests (1). Im Request wird ein Parameter mit dem Namen „action" und ein Parameter mit dem Namen „ausgabeformat" gesetzt. Alle Requests, die an das MCubeServlet geschickt werden, werden zuvor an eine so genannte Filterklasse weitergeleitet. Diese Klasse ist somit in der Lage, Requests zu verändern oder auch auszusortieren, bevor sie tatsächlich verarbeitet werden. Der Filter sieht im XML-Konfigurationsfile nach, zu welcher Applikation der Request gehört. Danach werden alle gültigen HTTP-Requests an das Servlet weitergeleitet, der Rest wird verworfen. Das Servlet sieht nun in einem xml-Konfigurationsfile nach (2) und findet einen Eintrag wie den folgenden:

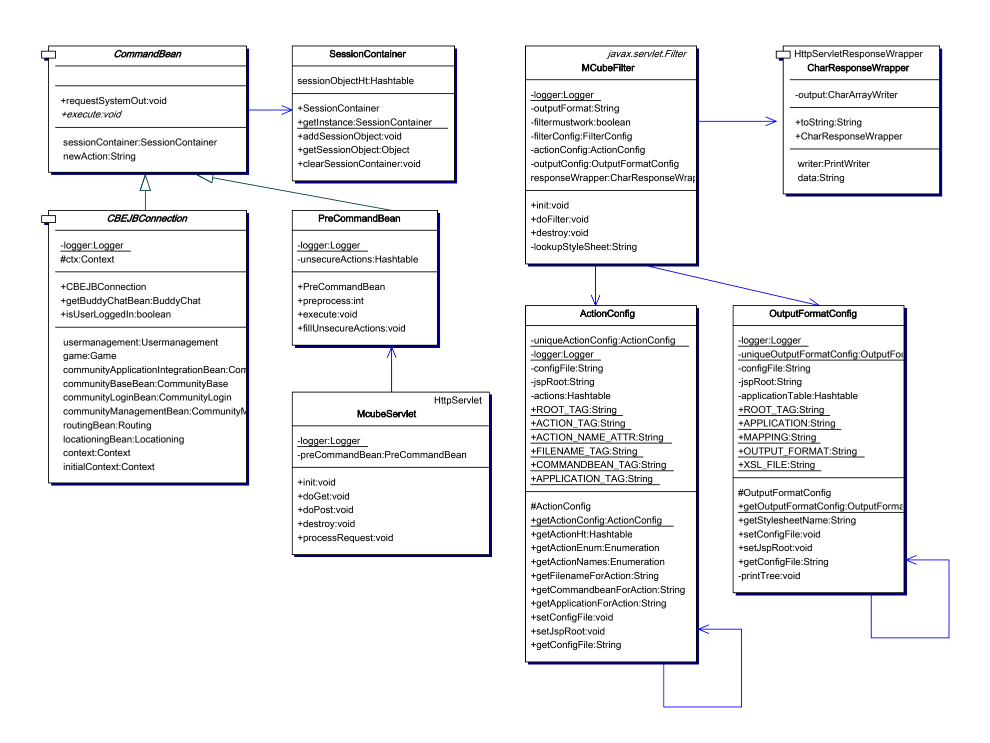


Abbildung 8.4: Klassendiagramm zum MVC-Konzept

```
<action name="start" unsecure="true">
  <application>community</application>
  <file>/start.jsp</file>
  <commandbean>
   mcube.application.community.command.CBStart
  </commandbean>
</action>
```

Steht im `unsecure`-Attribut der Wert „`false`“, so überprüft das Servlet, ob der Benutzer, der die Anfrage gestellt hat, momentan korrekt eingeloggt ist. Ist dies der Fall, initialisiert das Servlet ein entsprechendes CommandBean und ruft dessen execute-Methode auf (3), die jedes CommandBean besitzen muss. Der execute-Methode wird das Request und der Name der Action übergeben. Jetzt können im CommandBean Business-Logik ausgeführt und Werte in das Request geschrieben werden. Hier werden auch die EJBs angesprochen (4). Nachdem die execute-Methode ausgeführt ist, überprüft das Servlet, ob das CommandBean eine Nachfolge-Action gesetzt hat (3). Es ist also möglich, direkt auf die Bearbeitung einer Action eine weitere Action durchführen zu lassen. Ist eine Nachfolge-Action gesetzt, so holt sich das Servlet wieder das entsprechende CommandBean, initialisiert es und ruft dessen execute-Methode auf. Theoretisch kann das über sehr viele Stationen so laufen. Wenn irgendwann keine Nachfolge-Action mehr gesetzt sein sollte, sieht das Servlet im xml-Konfigurationsfile für die letzte aufgerufene Action nach (5), auf welche URL sie verzweigt und ruft diese auf (6). In der JSP werden nun eventuell Daten aus dem Request ausgelesen und in eventuell vorhandene Felder gesetzt. Daten für die JSPs können auch von den CommandBeans in den SessionContainer geschrieben werden (4b). Dieser dient zum Datenaustausch

von Daten, die für mehrere JSPs gelten. Außerdem ist noch zu sagen, dass eine Action nicht auf ein CommandBean zeigen muss. Ist der Eintrag für das CommandBean nicht vorhanden, wird direkt die für die Action angegebene URL aufgerufen. Die gefüllte JSP-Seite wird anschließend zum Browser zurückgeschickt (8a). Dieser Response enthält eine XML-Seite, die den Inhalt beschreibt und geräteunabhängig ist. Bevor der aufrufende Browser die Seite erhält (8c) wird sie wiederum vom Filter abgefangen, der eine XSLT-Transformation mittels eines ausgewählten Stylesheets (8b) durchführt. Das Stylesheet bestimmt sich aus der aktuellen Action und dem gewählten Ausgabeformat.

XSLT-Transformation: XSLT steht für Extensible Stylesheet Language for Transformations und bezeichnet mehrere Sprachen, mit deren Hilfe sich XML-Dateien transformieren lassen. XSLT besteht aus den Teilen XSL (Transformation von XML in XML), XPath (Beschreibung von Knoten- und Objektpositionen) sowie XSL-FO (Transformation in beliebige Formate)[Cla99].
Wir benötigen vor allem die Transformation in andere XML-Dateien bzw. XML sehr ähnlichen Formaten wie HTML und nutzen daher XSL und XPath. XSL ist eine funktionale Programmiersprache. Es können Transformationsregeln definiert werden, die auf bestimmte Knoten der XML-Eingabe angewendet werden. Diese Transformationsregeln werden in so genannten Stylesheetdateien abgelegt.
Im Projekt mCube existiert für jedes Ausgabeformat und jede Anwendung ein separates Stylesheet. Bevor der Request zum Browser zurückgesendet wird, ermittelt der Filter anhand der zuvor aufgerufenen Action und des als Parameter übergebenen Ausgabeformates das entsprechende Stylesheet, startet einen XSLT-Parser und sendet das Ergebnis an den Client zurück.
Als XSLT-Parser setzen wir Xalan ein, eine freie Implementierung des Apache-Projektes [Apa].

Was muss getan werden, um eine Applikation in das Konzept einzubinden? Man definiert bestimmte logische Actions, die aus dem Anwendungsfall klar werden. Eine Action wäre zum Beispiel login. Im Request werden der Action-Name (login) und außerdem die zwei Strings UserID und Password übergeben. Am Servlet muss nichts verändert werden. In das xml-Konfigurationsfile macht man nun zwei Einträge:

```
<action name="login" unsecure="true">
        <application>NewApp</application>
        <file>/start.jsp</file>
        <commandbean>
         mcube.application.community.command.CBLogIn
        </commandbean>
</action>

<action name="loginfailed" unsecure="true">
        <application>NewApp</application>
        <file>/loginfailed.htm</file>
</action>
```

Zusätzlich müssen noch die Attribute „unsecure“ und <application> definiert werden. In <application> steht der Name der zur Action gehörenden Anwendung, dies ist für das Auffinden eines passenden Stylesheets notwendig. „unsecure“ enthält den

Wert „true“, falls nicht überprüft werden soll, ob der anfragende Benutzer korrekt eingeloggt ist, ansonsten „false“.
Das Servlet ruft nun das CommandBean CBLogIn auf und startet die execute-Methode. Ein Entwickler muss also das CommandBean implementieren. In diesem Fall wird Zugriff auf eine EJB benötigt, daher erbt CBLogIn von CBEJBConnection. In dieser Oberklasse ist der Zugriff auf EJBs schon vorimplementiert. Werden weitere EJBs benötigt, die noch nicht in einer entsprechenden get-Methode zugreifbar sind, so sollte man CBEJBConnection so erweitern, dass auch die neue EJB ansprechbar ist. In der execute-Methode überprüft man per Zugriff auf das Usermanagement (oder die Community), ob der Username und das Passwort zueinander passen. Passen sie zueinander, ist man fertig (man könnte in die Session schreiben, wer jetzt eingeloggt ist, aber das ist für dieses Beispiel nicht wichtig). Das Servlet sieht nun nach, welche JSP zur Action gehört und verzweigt dann auf die Seite start.jsp. Nehmen wir nun an, das Passwort und der Username passen nicht zueinander. Man könnte nun einfach nextAction auf „loginfailed“ setzen. Das Servlet überprüft, ob das CommandBean eine nextAction gesetzt hat, in diesem Fall hat es das. Anstatt also start.jsp aufzurufen, wird die neue Action ausgeführt. Und da hier kein CommandBean angegeben ist, wird direkt auf die Seite loginfailed.htm verzweigt. Wenn ein CommandBean angegeben wäre, würde wieder die execute-Methode aufgerufen werden.

8.1.1.5 Package- und Verzeichnisstruktur

Die Komponenten des Projektes mCube sind auf verschiedene Packages aufgeteilt, um die Übersichtlichkeit zu erhöhen und um die logische Struktur der Architektur wiederzuspiegeln. Im Einzelnen ist folgende Struktur festgelegt worden:

- Die Komponenten der core-Schicht befinden sich im Package `mcube.core.xxx`, wobei `xxx` für den Namen der Komponente steht.
- Die Anwendungen befinden sich im Package `mcube.application.xxx`, wobei `xxx` für den Namen der Anwendung steht. Als Unterpackages existieren hier noch `mcube.application.xxx.component`, `mcube.application.xxx.gui` und `mcube.application.xxx.servlet` für die Model-, View- und Controller-Schichten.
- Das Package `mcube.util` beherbergt Hilfsklassen, die im gesamten Projekt relevant sind, also beispielsweise Klassen zum Austauschen von Daten. `mcube.util.interface` enthält Interfaces, die im gesamten Projekt relevant sind (natürlich nicht die EJB-Interfaces).

8.2 Die Anwendungen

8.2.1 mCube.community

8.2.1.1 Beschreibung

Die Community-Komponente bildet die grundlegende Anwendung der mCube-Plattform. Die von ihr dargestellte Grafik sieht der Nutzer bei jedem Login zuerst. Von ihr aus können weitere Anwendungen gestartet werden. Die Komponente bietet Basisdienste wie Chatten, FriendsAround und Administrationsdienste an. Die Integration der anderen Anwendungen soll sehr eng sein. Dies wird zum einen dadurch erreicht,

dass ein Teil der Oberfläche erhalten bleibt, egal, welche Anwendung gerade läuft. In diesem Bereich werden Statusmeldungen wie eingegangene Nachrichten und Navigationselemente angezeigt. Hier ist immer ein Wechsel zur Community möglich, notfalls wird eine laufende Anwendung zwangsweise unterbrochen.
Die Komponente ist in vier Bereiche aufgeteilt: Anwendungsintegration, Login, Verwaltung und Basisdienste. Alle Dienste können und sollen auch von den anderen Anwendungen genutzt werden. Manchmal ist es jedoch sinnvoller, direkt auf die Komponenten der core-Schicht zuzugreifen.
Abbildung 8.5 gibt einen Überblick über die Struktur der Komponente.

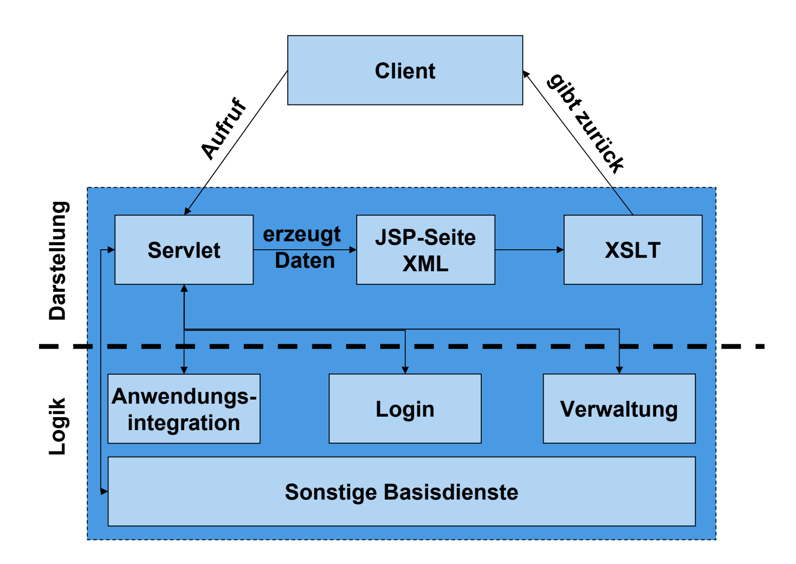


Abbildung 8.5: Struktur mCube.community

8.2.1.2 Anforderungen an die Komponente – API

Login

- **login(userID:String, password:String):boolean**

  User wird eingeloggt, Stati, die den Loginstatus betreffen, werden gesetzt. Dazu gehören auch Rechte für Attribute, die nur dann bearbeitet werden dürfen, wenn der Nutzer eingeloggt ist. True, wenn Login erfolgreich war, also insbesondere die `userID` vorhanden und das Passwort korrekt ist. Bei Rückgabe false wurde kein Login durchgeführt.

- **createNewUser(userID:String):boolean**

  Versucht, einen neuen Nutzer anzulegen. Falls der Name schon vergeben ist, wird `false` zurückgegeben. Sonstige Attribute werden hier noch nicht gesetzt. Das ist Aufgabe des Bereiches Verwaltung.

- **logout(userID:String):void**

  Der Nutzer wird ausgeloggt. Alle betreffenden Attribute werden gesetzt. Attribute, die nur bearbeitet werden dürfen, wenn der Nutzer eingeloggt ist, werden gesperrt.

- **userIsValidated(userID:String):void**

  Der User wurde authentifiziert. Gibt an, ob die bei der Anmeldung angegebenen Daten auch korrekt sind. Diese Überprüfung muss extern, beispielsweise über einen Adressvergleich erfolgen.

Verwaltung Methoden zur Verwaltung von Profilen und Objekten, sowohl für Nutzer als auch für Administratoren. Zu jedem Attribut des Profils gibt es eine `getXXX`- und eine `setXXX`-Methode. Diese überprüfen, ob die entsprechende Option gerade gelesen/gesetzt werden darf und führen die Aktion dann aus. Durch einzelne Methoden ist der Zugriff auf die Standard-Attribute einfacher und unkomplizierter.

- **userIsLoggedIn(userID:String):boolean**

  Gibt zurück, ob ein Nutzer gerade eingeloggt ist.

- **givePositioning(userID:String, begin:Date, end:Date):LinkedList**

  Gibt eine Liste mit Loggingobjekten zurück, in denen gespeichert ist, wann die Position des Nutzers bestimmt wurde. `begin` und `end` geben den Zeitraum an, für den die Daten geliefert werden sollen.

- **getAllBuddies(userID:String):LinkedList**

  getAllMessages(userID:String, componentID:String):LinkedList

  Liefert jeweils eine Liste mit allen speziellen Objekten eines Nutzers. Die Definition eines Filters ist somit nicht nötig.

- **addBuddy(userID:String, buddy:String):void**

  Fügt ein Objekt zur Buddyliste hinzu.

Basisdienste Die Basisdienste „Chat“ und „FriendsAround“ sind bereits in den entsprechenden Komponenten realisiert und werden hier nicht zusätzlich aufgeführt. Eine nochmalige Kapselung wäre eine Verschwendung von Zeit und Resourcen. Lediglich zusätzlich notwendige oder leicht abgeänderte Methoden werden hier zur Verfügung gestellt.

- **sendMessage(userID:String, message:Message):void**

  Sendet eine Nachricht an einen anderen User, der durch seine ID (seinen Login-Namen) gegeben ist.

Anwendungsintegration

- **displayMyMessages(componentID:String, iconID:String):void**

  Zeigt das Icon der Anwendung mit der ID `componentID` in der Statusleiste der GUI an, wenn eine neue Nachricht an diese Komponente für den Nutzer vorliegt. `iconID` ist die ID eines Bildobjektes, das in der Objektdatenbank gespeichert ist.

- **dontDisplayMyMessages(componentID:String):void**

  Es wird kein Icon mehr angezeigt, wenn eine Nachricht an diese Komponente für den Nutzer vorliegt.

- **isIconShown(componentID:String):boolean**

  Gibt zurück, ob die Komponente will, dass ein Icon bei Nachrichten angezeigt wird.

- **getIconsToBeShown(userID:String):LinkedList**

  Gibt eine Liste mit den ID's der Komponenten zurück, für die für den aktuellen Nutzer Nachrichten vorliegen. Eine Komponente ist natürlich nur dann in der Liste, wenn sie auch möchte, dass ihr Icon angezeigt wird.

- **swapApplicationOut(userID:String, componentID:String, state:Object):void**

  Diese Methode versucht, eine laufende Anwendung zu unterbrechen, ohne deren Zustand zu verlieren. Alle Anwendungen müssen eine Methode `swapApplicationOut(userID:String):Object` implementieren, die von der Community-Komponente aufgerufen wird. Ziel ist es, der Anwendung mitzuteilen, dass sie keine Aktionen für den Nutzer ausführen kann, bis sie wieder aktiv geschaltet wird. Das Objekt `state` und das von der Komponente zurückgegebene Objekt werden gespeichert und beim Reaktivieren der Komponente wieder übergeben. Gleichzeitig kann die Community-Komponente ihren Zustand speichern, um bei der Reaktivierung den alten Zustand wieder herstellen zu können.

  Die Methode ist eine Chance für die Komponente, sich auf die Deaktivierung einzustellen. Die Community-Komponente übernimmt auf jeden Fall die Vorherrschaft und zeichnet ihre GUI. Dieser Mechanismus ist voraussichtlich hochkompliziert im Einsatz und sollte nicht über mehrere Ebenen hinweg genutzt werden.

- **swapApplicationIn(userID:String, componentID:String):Object**

  Einer angehaltenen Anwendung wird mitgeteilt, dass sie wieder laufen kann. Diese Komponente muss dazu die Methode `swapApplicationIn(userID:String, state:Object):void` implementieren. Ihr wird mitgeteilt, dass sie wieder Aktionen mit dem Nutzer durchführen kann. Ein eventuell übergebenes Objekt wird ihr wieder zurückgegeben.

8.2.1.3 Interner Aufbau der Komponente

Es existieren vier SessionBeans, je eines für die Bereiche Login, Verwaltung, Basisdienste und Anwendungsintegration. Auf diese Beans greifen ein Servlet und/oder JSP-Seiten zu. Diese Beans nutzen die Methoden der Fassaden und implementieren darauf aufbauend weitere für die Komponente notwendige Logik. Daneben existieren Hilfsklassen, die zum Austausch von Daten benötigt werden. Eine Übersicht über den Aufbau gibt das UML-Klassendiagramm in Abbildung 8.6.

UML-Klassendiagramm Die Struktur der Anwendung mCube.community ist in Abbildung 8.6 grafisch dargestellt.

8.2.1.4 Anforderungen an andere Komponenten

Die Community-Komponente nutzt fast alle Bereiche von mCube.core.

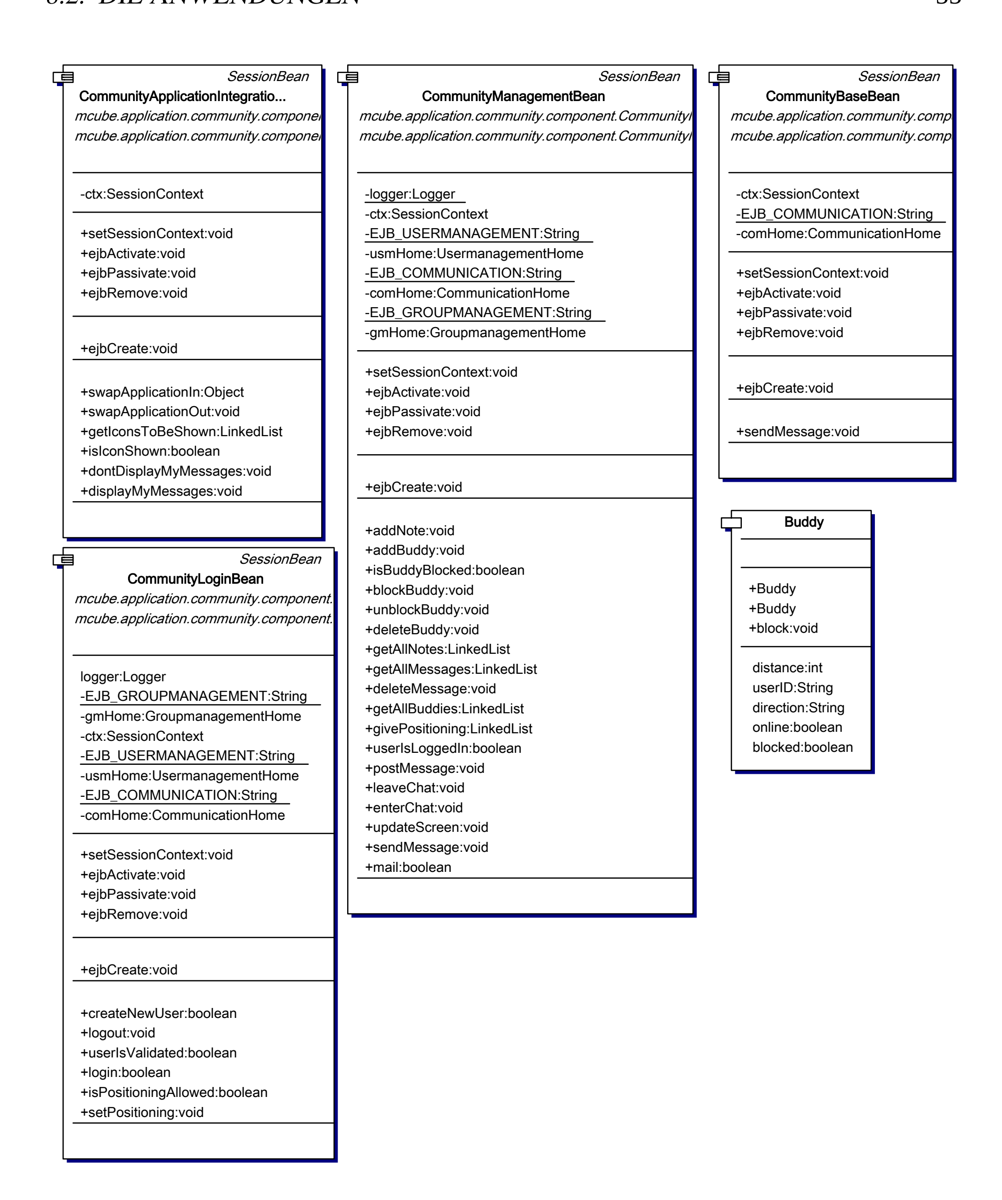


Abbildung 8.6: UML-Klassendiagramm der Komponente .community

8.2.2 mCube.dating

8.2.2.1 Beschreibung und Workflow

Diese Komponente setzt auf der Community-Plattform mCube auf. Die benötigten Funktionen werden durch den Zugriff auf die Fassade, die auch Grundlage der Community ist, realisiert

Mit der Absicht jemand anderen zu treffen, können Such-Profile erstellt werden, wonach unter allen mCube.dating Usern gesucht werden kann. Dabei wird das passendste Profil in der näheren Umgebung gesucht.

Man kann auf diese Weise User aussuchen und in einer eigenen Buddyliste speichern

und kontaktieren. Das geschieht über eine Nachricht (asynchron) oder über einen Chat (synchron). Um sich mit diesem User zu treffen, können Orte in der Umgebung gesucht werden.
Im Folgenden werden die verschiedenen Workflows zu den möglichen Funktionen beschrieben.
Einige der unten aufgelisteten Methoden können fast unverändert an die .core Komponenten weitergeleitet werden.

Login – Registrierung Um sich bei einer, auf der Community basierenden, Anwendung einzuloggen, muss man notwendigerweise einen Account in der Community haben. Diese Accountdaten werden als Default-Werte für den Login bei mCube.dating verwendet, können aber vom User bei der Registrierung geändert werden. Beim fehlgeschlagenen Login erhält der User die Aufforderung zur Registrierung.
Bei der erstmaligen Registrierung muss der User ergänzende Profildaten zum Community-Profil eingeben. Diese werden unter dem Punkt 1.2 Attribute genauer beschrieben.

Suche Die Suche ist in zwei Bereiche aufgeteilt. Man kann nach einem Partner suchen oder einem Ort.
Bei der Partnersuche werden aus vorgegebenen Variationen von Attributen Partnerprofile ausgewählt. Zum Beispiel kann bei der Auswahl des Alters ein Bereich vom User definiert werden, indem er das Maximal- und Minimalalter angibt. Bei fehlender Eingabe wird dieses Attribut bei der Suche außer acht gelassen und nur nach Übereinstimmung mit den restlichen Attributen gesucht. Bei Ausführung werden die 10 besten passenden User auf das Suchprofil aufgelistet, wobei nur User betrachtet werden, die Mitglied bei mCube.dating sind, die Ihre Positionsermittlung erlauben und die sich in dem eingestellten Suchradius befinden (dieser wird bei Optionen eingestellt). Die ermittelten User können zu der Buddyliste hinzugefügt werden.
Die Suche nach einem Ort ermittelt die Orte aus der .dating Sammlung, die möglichst günstig zwischen den beiden Usern liegen. Wenn bei der Suche nach einem Ort das Date-Partner Attribut nicht gesetzt ist, kann die Suche nicht durchgeführt werden.

Liste In diesem Bereich kann man zwei Listen aufrufen. Einmal die globale Userliste von .dating und einmal die Buddyliste des aktiven Users. Da für .dating je User ein neues Unterprofil erzeugt wird, ist es einfach, diese beim Usermanagement zu erfragen und eine Liste aller User anzuzeigen. Funktionen im Zusammenhang mit der globalen Liste wären: Profildetails aufrufen, online/offline, als Partner setzen, zu Buddyliste hinzufügen, sperren oder entsperren.
Die Buddyliste wird durch die FriendsAround-Komponente realisiert. User in der Buddyliste stellen folgende Funktionen bereit: Profildetails aufrufen, online/offline, als Partner setzen, sperren, entsperren und löschen.

Message Es ist möglich, eine Nachricht (asynchron) an den Partner zu senden oder mit ihm einen Chat (synchron) zu eröffnen. Kommuniziert werden kann mit allen Usern über die Community, hier jedoch muss immer das Attribut *Partner* mit dem User gesetzt sein, der als Partner ausgewählt wurde. Der Chat wird ebenfalls nur mit dem Partner geführt, es sei denn man hat ein Gruppendating definiert und der Partner ist eine Gruppe. In diesem Fall kann der Chat mit allen Gruppenmitgliedern der Gruppen durchgeführt werden.

Date-Einladungen, Nachrichten usw. werden in einem Posteingang (Queue von Kommunikation-Komponente) gesammelt und können abgerufen werden. Sie werden dann lokal in der Anwendung gespeichert.

Optionen Es können Attribute gesetzt oder geändert werden. Zum Beispiel kann die Ermittlung der eigenen Position verboten werden, der Radius des Suchbereichs gesetzt werden oder das eigene Profil editiert werden.

8.2.2.2 Attribute

Die Attribute erhalten alle Setter- und Getter-Methoden, die allerdings nicht in die API aufgenommen werden.

- **userID:String**

  Wird als Defaultwert aus der Community übernommen, kann aber bei der Erstellung des Profils geändert werden. Diese muss bei der Anwendung mCube.dating eindeutig sein und kann später nicht geändert werden.

- **password:String**

  Das Passwort wird initial ebenfalls aus der Community übernommen, kann aber auch bei der Registrierung oder im Menu Profiländerung geändert werden.

- **height:int**

  Größe des Users. (Angabe in cm)

- **haircolor:String**

  Haarfarbe des Users.

- **smoker:boolean**

  Angabe, ob der User Raucher oder Nichtraucher ist.

- **partner:String**

  Es kann zu jedem Zeitpunkt immer nur ein Partner gesetzt sein. Dieser wird benötigt, damit viele Funktionen, die sich um den Partner drehen, leichter benutzt werden können; zum Beispiel, um einen Ort zwischen zwei Usern herauszufinden oder dem Partner eine Nachricht zukommen zu lassen.

- **blockedUsers:LinkedList**

  Diese Liste enthält alle UserIDs, die durch den User gesperrt wurden.

- **searchRadius:int**

  Der Radius um die eigene Position, in der gesucht werden soll. (Angabe in Metern)

- **messages:LinkedList**

  Hier werden die empfangenen Nachrichten gespeichert.

8.2.2.3 Anforderungen an die Komponente – API

UML-Klassendiagramm Die Struktur der Anwendung mCube.dating ist in Abbildung 8.7 grafisch dargestellt.

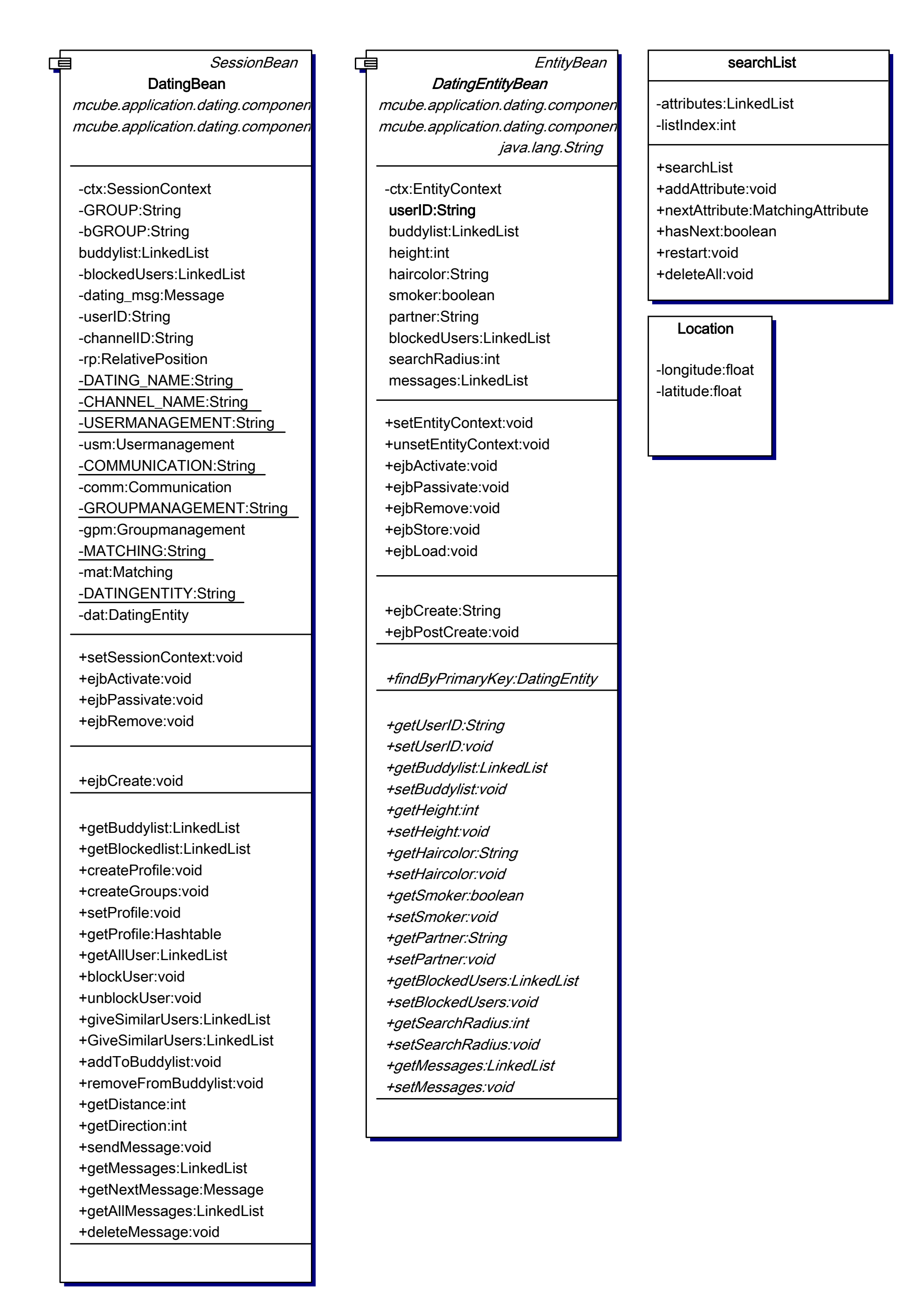


Abbildung 8.7: UML-Klassendiagramm der Dating-Komponente

Methoden

- **logIn(userID:String, password:String):boolean**

  Zum Anmelden bei mCube.dating. Nach der Identifikation können alle Attribute

des Profils gesetzt werden.

- **logOut():void**

  Der User wird von mCube.dating abgemeldet und wechselt zurück in die Community.

- **createProfile(userProfile:HashTable):void**

  Erstellt ein Userprofil für mCube.dating, das ein Subprofil der Community ist. Die Attribute werden in einer HashTable übergeben.

- **getProfile(userID:String):HashTable**

  Liefert das Profil eines Users zurück, als HashTable von Attributen. Daraus kann das (die) benötigte(n) Attribut(e) ausgewählt werden.

- **getPassword(userID:String):void**

  Sendet das .dating Passwort als Email an den User.

- **getAllUser():ListOfUser**

  Liefert alle User zurück, die sich in dem eingestellten Bereich befinden.

- **blockUser(userID:String):void**

  Sperrt einen User. Einladungen und jegliche Kontaktaufnahmeversuche durch diesen User werden automatisch abgelehnt.

- **unblockUser(userID:String):void**

  Ein gesperrter User kann entsperrt werden.

- **giveSimilarUsers(userProfile:HashTable):ListOfUsers**

  Benutzt die Methode der Matching-Komponente. Hier muss die Kooperation zu der Matching-Komponente noch genauer ausgearbeitet werden.

- **getDistance(UserID:String):int**

  Gibt die Entfernung in Metern zu einem anderen User zurück.

- **getDirection(UserID:String):int**

  Liefert die Himmelsrichtung zurück, in der sich der andere User befindet, relativ zu der Eigenen.

- **addToBuddylist(userID:String):void**

  Fügt einen User in die Buddyliste ein.

- **removeFromBuddylist(userID:String):void**

  Löscht einen User aus der Buddylist.

- **sendMessage(userID:String, msg:Object):void**

  Sendet eine Nachricht an den durch `userID` identifizierten User.

- **getMessages():ListOfMessages**

  Holt die Nachrichten für die Mailbox bei der Communication-Komponente ab. Diese können anschließend in der Anwendung lokal gespeichert werden.

- **openChat(groupID:String):boolean**
  Öffnet einen Chat mit einer Gruppe, die auch aus nur einem User bestehen kann.
- **closeChat():boolean**
  Schließt den aktuellen Chat. Es kann immer nur ein geöffneter Chat bestehen.
- **setPartner(userID:String):void**
  Setzt das Attribut Partner auf einen User. Dieser wird benötigt, um z.B. eine Location in der Mitte der beiden User zu berechnen.
- **invitePerson(email:String):void**
  Eine Einladung zu einer Person senden, die noch nicht Mitglied des mCube.dating Service ist.
- **searchLocation(ListOfProperties):ListOfLocationIDs**
  Sucht mögliche Locations (Treffpunkte) zwischen zwei Usern. Der eine User ist der gerade aktive, der andere ist das Attribut Partner. Dazu wird die Methode *meetingPlace* der Routing-Komponente benutzt.
- **getLocation(LocationID:String):Location**
  Liefert das Profil einer Location zurück, d.h. alle Detaildaten, die über diese Location gespeichert sind.
- **confirmMeeting(code:String):boolean**
  Erwartet einen Code des Date-Partners, womit das Treffen bestätigt wird.

Hilfsklassen

- **Locations**
  Da die Objektverwaltung von jeder Anwendung selbst durchgeführt wird, gibt es die Klasse Locations, in der alle Orte, die mit .dating gefunden werden können, gespeichert werden. Die Klasse hat noch eine Unterklasse, in der die Detaildaten zu jedem Ort gespeichert sind.
  - **locations:List**
  - **add(locationID:String, locationProfile:HashTable):void**
    Fügt einen Ort hinzu. Nur mit Admin-Rechten durchzuführen!
  - **removeFromList(locationID:String):void**
    Löscht einen Ort. Nur mit Admin-Rechten durchzuführen!
  - **getAllOfType(type:String):ListOfLocationIDs**
    Gibt alle Orte eines bestimmten Typs zurück. (Bsp.: Kino)
    - **interne Klasse: Location**
    - **locationID:String**
    - **coordinates:int×int**
    - **type:String**
    - **url:Url**
    - **phone:int**
    - **getProfile(locationID:String):HashTable**
      Liefert eine Hashtable mit den Attributen des Ortes zurück.

8.2.2.4 Möglicher interner Aufbau der Komponente

Eine EntityBean mit vorgeschalteter SessionBean, worauf die Servlets oder JSP-Seiten zugreifen. Viele Methoden werden für .dating leicht abgewandelt, können dann aber direkt zu der Fassade weitergeleitet werden. Weitere Methoden implementieren die Logik der .dating Anwendung. Hierzu werden Hilfsklassen benötigt, um zum Beispiel das Objektmanagement zu realisieren.

8.2.2.5 Anforderungen an andere Komponenten

- **User- und Gruppenmanagement**

  Es müssen User und Gruppen erstellt werden können und die Profile (Detaildaten) müssen generell von allen .dating Usern abgefragt werden dürfen.

- **Communication-Komponente**

  Es müssen Nachrichten gesendet werden können, die eine einfache Textbotschaft sein kann, aber auch eine Multimedia Message, ein Treffpunkt oder eine Einladung zum Date. Diese werden als Java-Objekt an die Communication-Komponente übergeben. So sollte auch abgefragt werden können, ob neue Nachrichten in der Mailbox bereit liegen.

- **Routing-Komponente**

  Es müssen Treffpunkte zwischen Usern abgefragt und angezeigt werden können. Da die Routing-Komponente auf die Location-Komponente zugreift, wird sie gar nicht benutzt, da nur Abstände zwischen zwei Usern benötigt werden.

- **FriendsAround-Komponente**

  Um die Buddyliste zu realisieren, sind sämtliche Funktionen des FriendsAround Dienstes nötig.

- **Matching-Komponente**

  Es werden die Funktionen benötigt, um die Ähnlichkeit zwischen zwei Userprofilen zu errechnen. Ebenso die Ähnlichkeit zwischen zwei Gruppen, die aus einer Menge von Usern bestehen.

8.2.3 mCube.game

mCube.game ist eine der zwei Beispielanwendungen, die für mCube.community erstellt wurde. Die Spielregeln sind ausführlich im Anforderungskapitel beschrieben und werden daher nicht in dieses Kapitel übernommen. Diese Anwendung besteht aus zwei Komponenten: GameEngine und Game. Auf Game gehen wir in diesem Kapitel ein, auf GameEngine im nächsten.

8.2.3.1 Beschreibung

Game ist eine stateless SessionBean, die außer der dem Ausführen eines Kampfes alle Funktionalität enthält. Die Kampf-Funktionalität befindet sich in der Komponente GameEngine, die in Abschnitt 8.2.4 beschrieben wird.
Im Folgenden stehe Charakter für die Spielfigur dieses Spieles. Im Hauptbildschirm gibt es einen Link zur Charaktererstellung. Sobald man einen Charakter erstellt hat,

wird man der Gruppe der Game-User hinzugefügt. Wird der Charakter gelöscht, so wird der entsprechende Spieler wieder aus der Gruppe der Game-User entfernt. Im Hauptbildschirm werden alle eingetragenen Spieler mit Charakternamen, Level und Entfernung angezeigt.
Es gibt folgende Typen von Nachrichten, die man von dieser Seite an Spieler senden kann:

- TradeOffer
- BattleInvitation
- Standardnachricht

Typen von Nachrichten, die automatisch gesendet werden:

- BattleStart
- BattleResult

Sendet man eine Nachricht an einen Spieler, so wird diese so lange im Application Server gespeichert, bis der andere Benutzer sie abholt und bearbeitet. Um Nachrichten abzuholen gibt es im Hauptbildschirm einen Button "Nachricht abholen". Je nachdem was es für ein Typ von Nachricht ist, gelangt man auf eine andere JSP-Seite. Bei einer BattleInvitation gelangt man auf eine Seite auf der die Aufforderung des anderen Spielers steht und dann zwei Buttons: Annahme und Ablehnung. Drückt man auf Ablehnung, so erhält der andere Spieler eine Standardnachricht mit der Ablehnung. Drückt man auf Annahme, so wird eine Bean GameEngine erstellt. Weitere Informationen zum Ablauf des Kampfes siehe Beschreibung von mCube.game.gameEngine.
Wenn man ein TradeOffer verschickt, so gibt man darin ein Angebot ab. Man gibt also 0 bis n Karten an, die man selbst besitzt und fordert dafür 0 bis n Karten, die der andere Spieler besitzt. Erhält man ein TradeOffer, so gelangt man auf eine Seite, auf der der übermittelte Text steht, die zu tauschenden Karten und ein Button annehmen bzw. ablehnen. Drückt man auf Annehmen, so werden die Karten in den game-Profilen getauscht, und beide Spieler erhalten eine Standardnachricht, dass der Tausch funktioniert hat. Drückt man auf Ablehnen, so wird der Tausch nicht durchgeführt.
Es ist ebenfalls möglich, vom Hauptbildschirm in den Schirm „Charakter bearbeiten" zu wechseln. Hier ist es möglich, den Charakter zu löschen, außerdem kann man sich die Beschreibungen der Karten, die der Charakter besitzt, durchlesen. Desweiteren kann man hier die Basiskarten des Charakters setzen, das sind die Karten, die der Charakter standardmäßig für einen Kampf voreingestellt hat. Außerdem kann man hier die Steigerung in eine neue Stufe (siehe Spielregeln) durchführen, wenn die Punkte es erlauben. Dann wird man in einen Bildschirm geführt, in dem man die Grundeigenschaft auswählen kann, die man steigern möchte. Nach der Auswahl wird das Attribut numberOfImproves im game-Profil um eins gesenkt.
Des Weiteren gibt es vom Hauptbildschirm einen Link auf eine Seite, auf der man sich seine Historie ansehen kann, hier stehen kurze Beschreibungen der erfolgten Kämpfe.
Löscht man seinen Charakter, so wird man aus der Gruppe der Game-User entfernt.

Das Unterprofil mCube.game Folgende Daten stehen im Unterprofil mCube.game (Charakter steht hierbei für die Spielfigur):

- String characterName - Fantasiename des Charakters

- int level - Level des Charakters
- int experiencePoints - Erfahrungspunkte des Charakters
- int lifePoints - Lebenspunkte des Charakters
- List of Strings eventCards - Eventkarten des Charakters. Aus diesen kann sich der Charakter jede Runde eine aussuchen, diese ist dann im Kampf nicht mehr einsetzbar
- List of Strings weaponCards - Waffenkarten des Charakters, aus diesen kann sich der Charakter in jeder Kampfrunde eine aussuchen.
- List of Strings itemCards - Übrige Karten des Charakters, aus diesen kann sich der Charakter eine bestimmte Anzahl zum Anfang eines Kampfes aussuchen, die dann den ganzen Kampf über gelten. (Rüstungen, Amulette usw.)
- List of Strings usualCards - Diese Karten sind eine Untermenge von itemCards, die als Voreinstellung dienen (zum Beispiel: Ritterrüstung, Amulett, Zauberring)
- String characterType - Typ des Charakters (Elf, Magier, Barbar oder Paladin)
- int intelligence - Die Intelligenz des Charakters.
- int strength - Die Stärke des Charakters.
- int constitution - Die Konstitution des Charakters.
- int dexterity - Die Geschicklichkeit des Charakters.
- int numberOfImproves - Die Anzahl der Steigerungen, die ein Charakter noch ausführen darf.
- List of Strings battleHistory - Eine Historie der Kämpfe des Charakters, zumindest mit Gegner und Ausgang.
- int maxDistance - Die maximale Distanz, die ein anderer Online-Spieler noch haben kann, um in der Liste des Spielers angezeigt zu werden.

Die Speicherung der Karten Die Daten der Karten werden in Extratabellen gespeichert, die zu mCube.game und mCube.game.gameEngine gehören (sie werden in beiden Beans benötigt), so dass die Strings, die in den Listen stehen, eindeutige Referenzen auf die Karten bilden.

- Allgemeine Daten:
  - String cardName - Name der Karte, gleichzeitig eindeutiger Key.
  - String description - Eine Beschreibung des Gegenstands oder der Handlung.
  - String type - Der Typ der Karte, hier gibt es zunächst folgende: Weapon (dieser Typ von Karte kann pro Kampfrunde neu gewählt werden, es zählen auch Zaubersprüche dazu), Event (pro Kampfrunde kann eine Eventkarte eingesetzt werden), Armour (im Kampf darf hiervon eine getragen

werden), Neck (alle Dinge, die um den Hals getragen werden, wie Amulette, im Kampf darf eine Karte dieses Types getragen werden), Hand (alle Dinge, die an der Hand getragen werden, meistens Ringe, im Kampf dürfen zwei Karten dieses Types getragen werden), Belt (alle Dinge, die am Gürtel getragen werden können, wie zum Beispiel Fläschchen mit Heiltränken, im Kampf können zwei Karten dieses Typs getragen werden) und Unlimited. (das sind Karten, die in beliebiger Anzahl mit in den Kampf genommen werden können)

- Voraussetzungen, die der Charakter benötigt:
  - int minConstitution - Die minimale Konstitution, die der Charakter benötigt, um die Karte einzusetzen.
  - int maxConstitution - Die maximale Konstitution, die der Charakter haben darf, um die Karte einzusetzen.
  - int minStrength - Die minimale Stärke, die der Charakter benötigt, um die Karte einzusetzen.
  - int maxStrength - Die maximale Stärke, die der Charakter haben darf, um die Karte einzusetzen.
  - int minIntelligence - Die minimale Intelligenz, die der Charakter benötigt, um die Karte einzusetzen.
  - int maxIntelligence - Die maximale Intelligenz, die der Charakter haben darf, um die Karte einzusetzen.
  - int minDexterity - Die minimale Geschicklichkeit, die der Charakter benötigt, um die Karte einzusetzen.
  - int maxDexterity - Die maximale Geschicklichkeit, die der Charakter haben darf, um die Karte einzusetzen.
  - List of Strings allowedCharacterTypes - Hier stehen die erlaubten Charaktertypen, also die Typen, die die Karte einsetzen dürfen.
- Modifikationen, die von der Karte ausgelöst werden:
  - int offensiveBonus - Der Offensivbonus der Karte, der für den benutzenden Charakter gilt, hauptsächlich benötigt für Karten vom Typ Weapon, allerdings können auch andere Karten den Offensivbonus eines Charakters verändern.
  - int defensiveBonus - Der Defensivbonus der Karte, der für den benutzenden Charakter gilt, hauptsächlich benötigt für Karten vom Typ Weapon, allerdings können auch andere Karten den Defensivbonus eines Charakters verändern (zum Beispiel eine Rüstung).
  - int offensiveBonusForOpponent - Der Offensivbonus der Karte, der für den Gegner gilt.
  - int defensiveBonusForOpponent - Der Defensivbonus der Karte, der für den Gegner gilt.
  - int additionalDamage - Die Anzahl der Lebenspunkte, die der Gegner verliert. Durch einen negativen Wert erhält der Gegner Lebenspunkte hinzu.

  - int additionalLifepoints - Die Anzahl der Lebenspunkte, die der benutzende Charakter erhält. Durch dieses Attribut können auch Schadenspunkte an den benutzenden Charakter verübt werden, indem ein negativer Wert eingestellt wird.
  - boolean noDamageForUsingCharacter - Falls auf true, gibt es in der Runde keine Schadenspunkte für den benutzenden Charakter.
  - boolean noDamageForOpponent - Falls auf true, gibt es in der Runde keine Schadenspunkte für den Gegner.
  - int modConstitution - Die Veränderung, die die Karte am Konstitutionswert des Charakters auslöst. Gilt für den benutzenden Spieler.
  - int modStrength - Die Veränderung, die die Karte am Stärkewert des Charakters auslöst. Gilt für den benutzenden Spieler.
  - int modIntelligence - Die Veränderung, die die Karte am Intelligenzwert des Charakters auslöst. Gilt für den benutzenden Spieler.
  - int modDexterity - Die Veränderung, die die Karte am Geschicklichkeitswert des Charakters auslöst. Gilt für den benutzenden Spieler.
  - int modConstitutionForOpponent - Die Veränderung, die die Karte am Konstitutionswert des Charakters auslöst. Gilt für den Gegner.
  - int modStrengthForOpponent - Die Veränderung, die die Karte am Stärkewert des Charakters auslöst. Gilt für den Gegner.
  - int modIntelligenceForOpponent - Die Veränderung, die die Karte am Intelligenzwert des Charakters auslöst. Gilt für den Gegner.
  - int modDexterityForOpponent - Die Veränderung, die die Karte am Geschicklichkeitswert des Charakters auslöst. Gilt für den Gegner.

- Spezialfälle:

  - String modificationsNextRound - Der Name zeigt auf einen weiteren Eintrag in der Tabelle. Die Modifikationen, die dieser Eintrag verursacht, werden allerdings erst in der nächsten Kampfrunde wirksam. So wird die Karte Ausweichen realisiert, indem dem Eintrag Ausweichen nur die beiden Attribute noDamageForUsingCharacter und noDamageForOpponent auf true stehen, alle anderen Attribute stehen auf 0 bzw. einem sinnvollen Wert, und unter dem Eintrag, der unter modificationsNextRound zu finden ist, steht nur offensiveBonus auf 5.
  - List of Strings dependencyAttributes - Hier stehen alle Attribute, die in eine Berechnung eines Kampfwertes eingehen, nur benötigt für Karten vom Typ Weapon.
  - int maxDamage - Der maximale Schaden der Karte, nur benötigt für Karten vom Typ Weapon.
  - int minDamage - Der minimale Schaden der Karte, nur benötigt für Karten vom Typ Weapon.

8.2.3.2 Anforderungen an die Komponente – API

- **getAttribute(userID:String, attribute:String):Object**

  Ein Attribut des Users wird ausgelesen und zurückgegeben. In dieser Komponente werden hauptsächlich Daten aus dem Game-Profil benötigt, wie zum Beispiel Charaktername, Level usw.

- **addPlayerToOnlinegroup (userID:String):void**

  Startet jemand diese Methode, so wird er automatisch zu den Online-Playern hinzugefügt. Ab jetzt wird er bei den anderen Spielern angezeigt.

- **removePlayerFromOnlinegroup (userID:String):void**

  Wird die Applikation beendet, wird er wieder entfernt.

- **getDistance(userID1:String, userID2:String):int**

  Gibt die Entfernung zwischen zwei Usern zurück. Wird benötigt für die Anzeige im Hauptbildschirm.

- **getDirection(user1ID:String, user2ID:String):String**

  Gibt die Richtung zurück, in die user2 im Verhältnis zu User1 steht. Wird benötigt für die Anzeige im Hauptbildschirm.

- **sendBattleInvitation(fromUserID:String, toUserID:String, slogan:String):void**

  Sendet eine Kampfaufforderung. Erläuterung siehe Beschreibung. slogan ist dabei eine Textnachricht an den aufgeforderten Spieler geschrieben vom Herausforderer.

- **sendTradeOffer(fromUserID:String, toUserID:String, slogan:String, offeredCards:List of Strings, wantedCards:List of Strings):void**

  Sendet ein Tauschangebot. Erläuterung siehe Beschreibung.

- **sendMessage (fromUserID:String, toUserID:String, slogan:String):void**

  Sendet eine Standardtextnachricht.

- **addUsualCard (userID:String, card:String):void**

  Zu den Basiskarten des Charakters des übergebenen Benutzers wird die übergebene Karte hinzugefügt.

- **removeUsualCard (userID:String, card:String):void**

  Aus den Basiskarten des Charakters des übergebenen Benutzers wird die übergebene Karte entfernt.

- **tradeItems (user1ID:String, user2ID:String, itemsOfUser1:List of Strings, itemsOfUser2:List of Strings):void**

  Diese Methode tauscht die angegebenen Items zwischen den Usern.

- **deleteCharacter (userID:String): void**

  Der Charakter des Users wird gelöscht. Dazu wird das Game-Unterprofil entfernt.

- **createCharacter(userID:String, typeOfCharacter:String, characterName:String):void**

  Der User erstellt einen neuen Charakter. Dazu wird ein Game-Unterprofil erstellt und das alte gegebenenfalls gelöscht.

- **removeForbiddenCards (userID:String, cards:List of Strings):List of Strings**

  Aus der übergebenen Liste werden alle Karten entfernt, die der Charakter des übergebenen Spielers nicht benutzen darf. Die erhaltene Liste wird zurückgegeben. Beim Entfernen der Karten werden folgende mögliche Einschränkungen beachtet: - Die Anzahl von tragbaren Karten eines Typs ist vorgegeben. So kann der Charakter nur eine Rüstung tragen, zwei Ringe und eine Kette. Sind mehr dieser Items in der Liste, so ist false zurückzugeben. - Die Karten können auf bestimmte Charaktertypen beschränkt sein, das ist auch zu überprüfen. - Die Karten können auf bestimmte Eigenschaftswerte beschränkt sein, zum Beispiel Stärke > 6, auch das ist zu überprüfen.

- **setMaxDistance (userID:String, int maxDistance):void**

  Mit dieser Methode kann die vom Spieler gewünschte maximale Distanz von Spielern eingestellt werden, die online sind.

- **setUsualCards (userID:String, usualCards:List of Strings):void**

  Mit dieser Methode können die Voreinstellungen für einen Kampf gesetzt werden. Hier übergibt man also Karten, deren Effekte alle gleichzeitig die Eigenschaften des Charakters verändern können. Um zu überprüfen, ob die Auswahl an Karten gleichzeitig von dem Charakter des Spielers genutzt werden kann, wird die folgende Methode benutzt.

- **isCardCombinationAllowed (userID:String, items:List of Strings):boolean**

  Diese Methode überprüft, ob die übergegebenen Karten gleichzeitig vom Charakter benutzt werden können. Dabei sind folgende mögliche Einschränkungen zu beachten: - Die Anzahl von tragbaren Karten eines Typs ist vorgegeben. So kann der Charakter nur eine Rüstung tragen, zwei Ringe und eine Kette. Sind mehr dieser Items in der Liste, so ist false zurückzugeben. - Die Karten können auf bestimmte Charaktertypen beschränkt sein, das ist auch zu überprüfen. - Die Karten können auf bestimmte Eigenschaftswerte beschränkt sein, zum Beispiel Stärke > 6, auch das ist zu überprüfen.

- **isCardAllowed (userID:String, card:String):boolean**

  Mit dieser Methode wird überprüft, ob der Charakter die übergebene Karte benutzen darf. Hier sind folgende Einschränkungen zu beachten: - Die Karte kann auf bestimmte Charaktertypen beschränkt sein, das ist zu überprüfen (Ritzkihutu nur für Elf). - Die Karte kann auf bestimmte Eigenschaftswerte beschränkt sein, zum Beispiel Stärke > 6, auch das ist zu überprüfen.

- **getNextMessage (userID:String):Message**

  Diese Methode holt die nächste Message für den User ab. Es werden hier nur Messages für Game abgeholt. Von der Art der Message hängt ab, in welchen Bildschirm die Applikation wechselt! Diese Methode nutzt die Methode getNextMessage (userID, application) aus der Kommunikationskomponente, wobei application hier gleich game ist.

- **getOnlineUsers (userID:String):List of PlayerDataForBuddyList**

  Diese Methode holt die Daten für den Startbildschirm. Das sind die UserIDs, die Charakternamen, die Level, Entfernungen und Richtungen aller Mitglieder der Gruppe Game-User mit Ausnahme des aktuellen Users. Diese Daten werden in Form einer Liste von PlayerDataForBuddyList-Objekten zurückgegeben.

- **Klasse PlayerDataForBuddyList**

  Diese Klasse kapselt die Daten, die auf der Startseite von Game benötigt werden. Dies sind:

  - int distance
  - String userID
  - String charactername
  - String direction
  - String level

- **Klasse TradeOffer**

  Diese Klasse erbt von Message und erhält die zusätzlichen Attribute:

  - List of Strings offeredCards
  - List of Strings wantedCards

  Das in Message vorhandene Feld content wird genutzt, um den Slogan unterzubringen. Sie wird in der Methode sendTradeOffer erstellt und versendet. Der Klassenname wird als Markierung benutzt, das ist wichtig für die Weiterverarbeitung.

- **Klasse BattleInvitation**

  Diese Klasse erbt von Message und erhält keine zusätzlichen Attribute. Der Klassenname wird als Markierung benutzt, das ist wichtig für die Weiterverarbeitung. Das in Message vorhandene Feld content wird genutzt, um den Slogan unterzubringen.

8.2.3.3 Möglicher interner Aufbau

Game ist eine Enterprise Java Bean vom Typ „Stateless SessionBean“. Dies ist möglich, da keine Daten in der Bean gehalten werden müssen. Bei jeder Anfrage kann also eine beliebige Bean aus dem Pool verwendet werden, was die Performanz gegenüber einer „Stateful SessionBean“ erheblich erhöht.
Card ist natürlich eine Enterprise Java Bean vom Typ „EntityBean“ mit „Container Managed Persistence“, das heißt, in ihr werden nur Daten gehalten, und die Verwaltung der Daten übernimmt der Container.

UML-Klassendiagramm Die Struktur der Komponente mCube.Game.Gameengine ist in Abbildung 8.8 grafisch dargestellt.

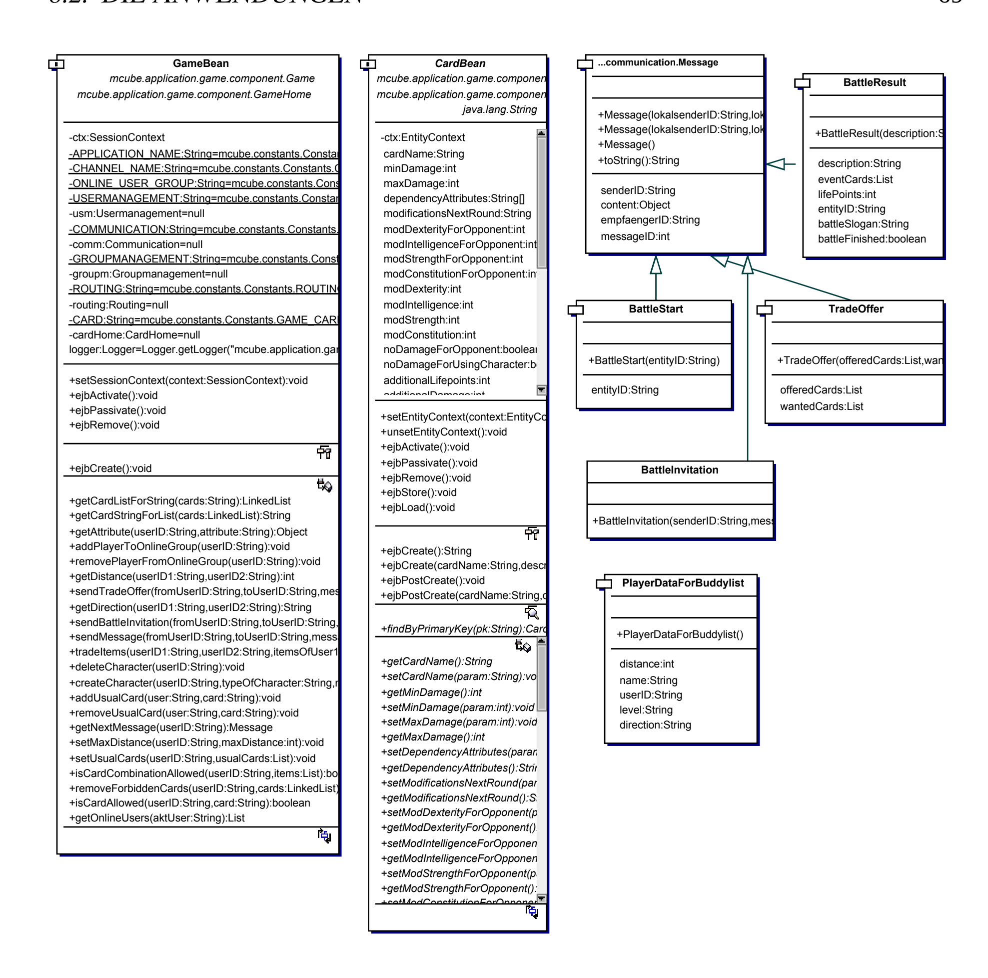


Abbildung 8.8: UML-Klassendiagramm der Komponente mCube.game

8.2.3.4 Anforderungen an andere Komponenten

- **User-/Rechtemanagement**

  Game muss die Möglichkeit haben, ein Datum eines bestimmten Users auszulesen; in diesem Fall die Daten aus dem Profil für Game.

- **Kommunikation**

  Game benötigt die Methode

  - **sendMessage (message, userID, application): void**
    um TradeOffers, BattleInvitations und Standard-Nachrichten an einen bestimmten User zu schicken, wobei application immer game ist.
  - **getNextMessage(userID, application):Message**
    wird benötigt, um Nachrichten abzurufen.

- **Routing**

  Es muss die Entfernung und Richtung zwischen zwei Usern herausgefunden werden.

- **Gruppenmanagement**

  Es muss möglich sein, ein User einer Gruppe hinzuzufügen und den User wieder aus der Gruppe zu löschen.

8.2.4 mCube.game.gameEngine

8.2.4.1 Beschreibung

Diese Komponente besteht aus einer EntityBean und einer vorgeschalteten SessionBean. Diese Komponente wird für einen Kampf zwischen zwei Usern erstellt. Sie beinhaltet für jeden der zwei User folgende Attribute:

- String userID
- int lifePoints
- List of Strings eventCards
- boolean gotTurnInfo
- TurnInfo turnInfo

Außerdem besitzt sie eine eindeutige ID. Diese Komponente ist für die Auswertung eines Kampfes verantwortlich. In ihrer create-Methode wird sie für zwei übergebene User initialisiert. Bei der Initialisierung geschieht Folgendes: Die Werte der Lebenspunkte und der Eventkarten werden aus dem Usermanagement in diese Komponente übernommen. Außerdem erhalten die Spieler eine Bestätigung, dass das Spiel jetzt losgeht, in Form einer Nachricht vom Typ BattleStart. Hier wird auch die eindeutige ID der EntityBean übergeben. Diese BattleStart-Nachricht führt die Spieler nun in den Startbildschirm des Kampfes, in dem beide Spieler noch einmal die Basiskarten angezeigt bekommen (also die Item-Karten, die man vor einstellen kann und die für den Kampf benutzt werden). Die Basiskarten können hier noch einmal verändert und so auf den aktuellen Gegner angepasst werden, dabei wird immer überprüft, ob die entsprechende Kartenkombination für den Charakter erlaubt ist.
Danach starten die normalen Kampfrunden, in denen jeweils eine Waffe ausgewählt wird, eventuell eine Event-Karte, einen Prozentwert für Offensivität / Defensivität und einen Kampfspruch. Dabei wird überprüft, ob die Eventkarte und die Waffenkarte benutzt werden dürfen. Diese Daten werden nun in einem Objekt vom Typ TurnInfo gekapselt. In jeder Kampfrunde wird ausgelöst durch jeden Spieler einmal die Methode battle(userID, turnInfo) aufgerufen. Die Methode speichert die turnInfo in der entsprechenden Spalte in der Datenbank und setzt gotTurnInfo für den entsprechenden User auf true. Wenn nun schon das turnInfo des anderen Spielers vorhanden ist, und der zweite Spieler ruft die Methode battle(userID, turnInfo) auf, dann wird der Zug ausgewertet.
Dazu werden zunächst die Basiswerte des Charakters durch die Modifikationen der Basiskarten verändert, dann wird der Kampf mit der entsprechenden Waffe wie im Anforderungskapitel erläutert durchgeführt. Dann werden die Lebenspunkte angepasst und in die Datenbank geschrieben, falls eine Eventkarte benutzt wurde, wird diese aus der Liste der Eventkarten gelöscht, damit sie in der nächsten Runde nicht mehr benutzt werden kann, und schließlich wird gotTurnInfo für beide wieder auf false gesetzt.
Am Ende werden Nachrichten vom Typ BattleResult an die beiden Spieler gesendet, in denen die wichtigen Daten, die sich verändert haben, übergeben werden. Dies sind

die Lebenspunkte und die neue Liste der Eventkarten, außerdem wird noch eine Beschreibung der Kampfrunde mitgeschickt und der Spruch, den der Gegner das letzte Mal gesagt hat. Wichtig ist auch die EntityID der mit dem Kampf verbundenen GameEngineEntity-Bean, damit diese für die Auswertung der nächsten Kampfrunde wiedergefunden werden kann. Falls einer der Spieler weniger als 0 Lebenspunkte haben sollte, wird die Variable isBattleFinished im BattleResult-Objekt auf true gesetzt. Außerdem werden dann die Erfahrungspunkte verteilt und eventuell die Steigerungsmöglichkeiten erhöht.
Um die richtige EntityBean wiederzufinden, gibt es in der SessionBean die Methode create (entityID), die abgesehen von der Initialisierung immer benutzt wird, um die Bean zu holen.

8.2.4.2 Anforderungen an die Komponente – API

- **create (userID1:String, userID2:String):komponente**

  Erzeugt einen Kampf zwischen den beiden übergebenen Usern. Das interne EntityBean wird erzeugt und wie oben beschrieben mit Daten gefüllt. Eine Nachricht vom Typ BattleStart wird an die beiden Kampfteilnehmer geschickt, wobei die eindeutige ID der erzeugten EntityBean mit übergeben wird.

- **create (entityID:int):komponente**

  Diese Methode wird benutzt, um die bereits initialisierte Bean zu holen. Dafür wird die entsprechende entityID verwendet, die in der BattleResult-Nachricht mit übergeben wird.

- **battle (TurnInfo turnInfo, userID:String):void**

  Die Zugdaten werden ins EntityBean übernommen und gotTurnInfo auf true gesetzt. Es wird überprüft, ob der andere gotTurnInfo-Wert ebenfalls true ist, falls ja, wird der Zug ausgewertet, es werden also alle Berechnungen angestellt, die Werte wie oben beschrieben aktualisiert und eine Nachricht vom Typ BattleResult an beide Teilnehmer geschickt.

- **Die Klasse BattleResult**

  Die Klasse BattleResult erbt von Message. Der Klassenname wird als Markierung benutzt, das ist wichtig für die Weiterverarbeitung. Sie beinhaltet folgende Attribute:

  - String description - Dieser String beinhaltet eine Beschreibung der vergangenen Kampfrunde.
  - Liste von Strings eventcards - Diese Liste beinhaltet alle Eventkarten, die im weiteren Spielverlauf noch benutzt werden können.
  - int lifepoints - Die Lebenspunkte, die man noch hat.
  - String entityID - Die ID der EntityBean dieser Komponente.
  - String battleSlogan - Der Slogan des anderen Spielers wird zur Anzeige mitgeschickt
  - boolean isBattleFinished - Zeigt an, ob der Kampf beendet ist oder fortgeführt werden muss

- **Die Klasse BattleStart**

  Die Klasse BattleStart erbt von Message. Der Klassenname wird als Markierung benutzt, das ist wichtig für die Weiterverarbeitung. Sie beinhaltet folgende Attribute:

  – String entityID - Die ID der EntityBean dieser Komponente.

- **Die Klasse TurnInfo**

  Die Klasse TurnInfo beschreibt einen Spielzug, den ein Spieler macht. Sie beinhaltet folgende Attribute:

  – int offensive - Die Offensivität des Angriffs in Prozent.
  – String event - Eine benutzte Eventkarte.
  – String weapon - Die benutzte Waffe. (kann auch Zauberspruch sein)
  – String slogan - Ein Spruch, der dem Gegner vor der Attacke gesagt wird.

8.2.4.3 Möglicher interner Aufbau

GameEngine ist eine Enterprise Java Bean vom Typ „Stateful SessionBean“. Sie muss stateful sein, da eine Referenz zu einer bestimmten GameEngine EntityBean gehalten werden muss.
GameEngineEntity ist eine Enterprise Java Bean vom Typ „EntityBean“ mit „Container Managed Persistence“, das heißt, in ihr werden nur Daten gehalten, und die Verwaltung der Daten übernimmt der Container.

UML-Klassendiagramm Obige Beschreibung ist in Abbildung 8.9 grafisch dargestellt.

8.2.4.4 Anforderungen an andere Komponenten

- **User-/Rechtemanagement**

  GameEngine muss die Möglichkeit haben, ein Datum eines bestimmten Users auszulesen, in diesem Fall die Daten aus dem Profil für Game.

- **Kommunikation**

  GameEngine benötigt die Methode

  – sendMessage (message, userID, application): void

  um Kampfauswertungen an einen bestimmten User zu schicken, wobei application immer game ist

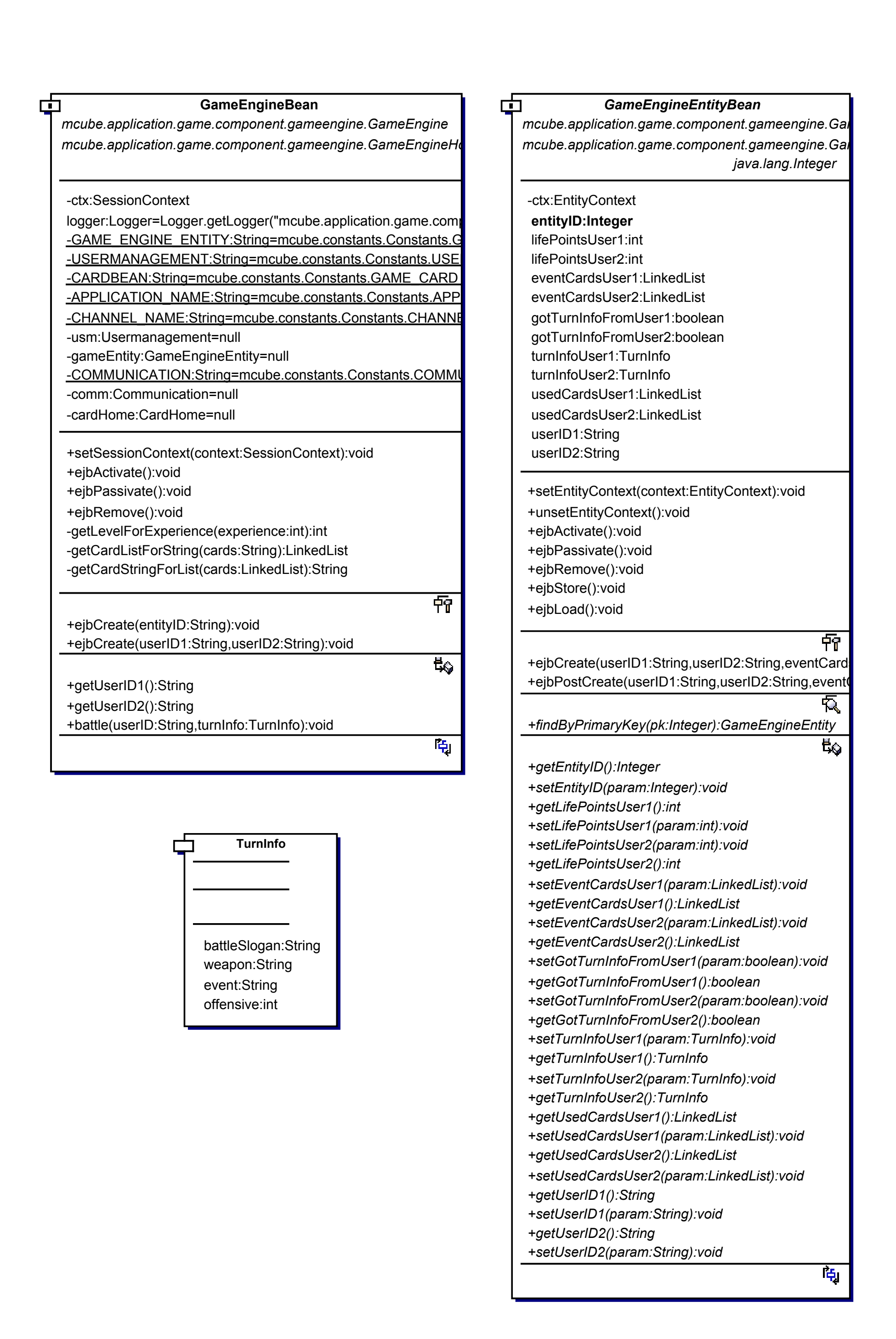


Abbildung 8.9: UML-Klassendiagramm der Komponente mCube.Game.GameEngine

Kapitel 9

Technische Infrastruktur

Dieses Kapitel liefert einen Überblick über die für den Betrieb von mCube nötige Infrastruktur. Die Architektur des Projekts mCube besteht aus 3 wesentlichen Teilen:

BEA Weblogic Application Server 7.0

Oracle Database 8*i*

Netscape Directory Server 6.0.1

9.1 Application Server

Zum Betrieb des Application Servers wurde eine JDBC-Connection zur Oracle Datenbank eingerichtet, wodurch der Application Server in der Lage ist, Java Beans persistent in der Oracle Datenbank abzulegen. Des Weiteren wurde ein JMS-Server (Java Messaging Service) [Sun03a] im Application Server eingerichtet, wodurch Nachrichtenversand innerhalb der Anwendung realisiert wird.

9.2 Oracle

Als Datenbank wurde eine Oracle 8 personal edition installiert, die vom Application Server benutzte Tabellen enthält. Um manuell auf die Datenbank zugreifen zu können, wurde das Tool „Toad“ benutzt. Folgende Informationen werden in der Oracle Datenbank gespeichert:

- Gruppenzugehörigkeiten der mCube-Benutzer
- Positions-Cache
- Zuordnung Postleitzahlen zu Positionen
- JMS-Nachrichten
- Daten der EntityBeans

9.3 Netscape Directory Server

Für die Verwendung eines Directory Servers haben wir uns entschieden, da von einem solchen Dienst bereits alle wichtigen Funktionen zur Benutzerverwaltung zur Verfügung gestellt werden und man über eine einheitliche Schnittstelle (LDAP) darauf zugreifen kann [Net03].
Ziel des Directory Servers ist es, alle Informationen eines Users - darunter fallen Benutzername, Vor- und Nachname und Passwort, aber auch alle mCube-spezifischen Daten - zu verwalten. Grundprinzip eines Directory-Servers ist ein Baum. Knoten dieses Baumes nennt man 'Container', Blätter 'Einträge' oder 'Objekte' und die Eigenschaften eines Blattes 'Attribute'. Objekte können von anderen Objekten (auch von mehreren) erben, sie gehören damit zu einer Menge von Klassen und besitzen somit alle Attribute dieser Menge. Die Zuordnung von Attributen zu einem bestimmten Objekt nennt man „Schema“. Beispielsweise sollte ein Objekt, welches die Daten eines Benutzers enthält, zu der Klasse „Person“ gehören und sich unterhalb des Containers „People“ befinden [IBM98]. Da Attribute nur zur bestimmten Objekten gehören können (man also nicht einem Objekt einer bestimmten Klasse Attribute zuordnen kann, die nicht in der Klasse enthalten sind), haben wir für das Projekt mCube eine eigene Klasse definiert: „mcubeprofil“. Diese Klasse erbt von der Klasse „person“ und hat zusätzlich folgende optionale Attribute:

- birthdate
- city
- cubies
- dating-haircolor
- dating-height
- dating-searchradius
- dating-smoker
- dating-weight
- email
- friendsaroundlist
- game-battlehistory
- game-charactername
- game-charactertype
- game-constitution
- game-dexterity
- game-eventcards
- game-experiencepoints
- game-intelligence

- game-itemcards
- game-level
- game-lifepoints
- game-maxdistance
- game-numberofimproves
- game-strength
- game-usualcards
- game-weaponcards
- isLoggedIn
- isValidated
- logintype
- plz
- positioningallowed
- sex
- street

Dies sind alle zurzeit für das Projekt mCube notwendigen Attribute. Jeder User im mCube-Projekt gehört daher zu folgenden Klassen:

- top
- person
- organizationalPerson
- inetOrgPerson
- mcubeprofile

Um zuordnen zu können, welche Attribute zu einem bestimmten Profil gehören (z.B. alle Attribute des Profils „Dating“), gibt es einen separaten Container „mCubeContainer“. Dieser enthält je ein Objekt der Klasse „mcubeprofiledefinition“ pro Profil. Diese Objekte enthalten ein Attribut „profilattributes“, in dem sich mehrere Einträge befinden, nämlich die Namen exakt der Attribute eines Users, die zu einem Profil gehören. Das sieht dann zum Beispiel wie folgt aus:

cn=Community,ou=MCubeContainer,dc=mcu,dc=be

- profilattributes: sn
- profilattributes: cn
- profilattributes: givenName
- profilattributes: mobile
- profilattributes: profiles
- profilattributes: birthdate
- profilattributes: city
- profilattributes: cubies
- profilattributes: description
- profilattributes: email
- profilattributes: friendsaroundlist
- profilattributes: isLoggedIn
- profilattributes: isValidated
- profilattributes: plz
- profilattributes: positioningallowed
- profilattributes: sex
- profilattributes: street
- profilattributes: telephoneNumber
- profilattributes: logintype
- objectClass: top
- objectClass: mcubeprofildefinition

Die oben aufgeführten Attribute „profilattributes“ sind also exakt die Attribute, die zum Profil „Community“ gehören. Eine weitere Besonderheit ist das Speichern von Floats, da der von uns verwendete Netscape Directory Server von Haus aus keinen Datentyp „Float“ kennt. Um dieses Problem zu umgehen, kann an jedes Userobjekt ein weiteres Objekt der Klasse „Floatcontainer“ gehängt werden. Der Name dieses Objekts entspricht dem Namen des zu speichernden Float-Attributes. Dieses Objekt hat dann ein Attribute „float“ , in dem sich der Float-Wert befindet.

Kapitel 10

Entwicklungsumgebung

Die Durchführung des Projektes der Erstellung einer Mobile Community Plattform erforderte weit reichenden Einsatz von Softwarewerkzeugen sowohl zur Planung der Entwicklung als auch bei der Durchführung der Entwicklungstätigkeit selbst. In diesem Kapitel werden wir die verwendete Entwicklungsumgebung der Projektgruppe vorstellen. Die Projektgruppe bekam vom Lehrstuhl 10 der Universität Dortmund einen Rechnerraum inklusive der zur Entwicklung notwendigen Hardware zur Verfügung gestellt, zu dem ausschließlich die Mitglieder der PG 409 und 411 Zugang hatten. Da das benötigte Entwicklungsszenario recht umfangreich war, konnte man es schlecht in einen öffentlichen Rechnerpool auslagern. Außerdem brachte diese Anordnung den Vorteil, ein einmal aufgebautes Entwicklungssystem weit gehend stabil zu halten, da es vor Veränderungen durch fremde Personen geschützt ist. Jede PG hat jeweils einen eigenen Server, die Entwicklungsrechner wurden jedoch gemeinschaftlich genutzt. Die gemeinsame Nutzung des Pools sollte auch dazu dienen, Erfahrungen auszutauschen, da beide PGs Bea Weblogic und eine Oracle Datenbank einsetzen. Im Folgenden wird die verwendete Software genau beschrieben.

10.1 Betriebssysteme

Für die Einrichtung einer Entwicklungsumgebung sowie für den Betrieb der eigentlichen Software musste eine Systemarchitektur entworfen und realisiert werden, die einerseits den stabilen Betrieb der mCube-Software inkl. aller dafür notwendigen Dienste wie Application Server, Datenbanken, Directory Services usw. gewährleistet, andererseits eine performante und sichere Entwicklungsumgebung für alle PG-Mitglieder zur Verfügung stellt. Um diese hohen Ansprüchen erfüllen zu können, wurde als Betriebssystem für den PG-Server ein Windows 2000 Server installiert und konfiguriert. Für die Arbeitsstationen wurde dementsprechend Windows 2000 Professional genutzt. Alle Rechner wurden über ein 100MBit-Netzwerk verbunden und der Server über eine Firewall geschützt.
Damit sich sowohl der Mitglieder der PG 409 als auch die Mitglieder der PG 411 an den Arbeitsstationen anmelden können und die Zugriffrechte klar definierbar sind, wurden 2 Domänen eingerichtet und beide PG-Server als PDC mit einem gegenseitigem Trust eingerichtet [Mic03]. Die jeweiligen User wurden dann in die entsprechende Domäne der PG gehangen und die Profile serverseitig gespeichert. Somit konnte man an jeder Arbeitsstation beim Anmelden die gewünschte Domäne wählen und sich mit seinem

Benutzernamen aus der Domäne anmelden. Das Profil des Users wurde dann automatisch vom Server geladen, wodurch ein arbeitsplatzunabhängiges Arbeiten gegeben war.
Für die PG-Mitglieder wurde in der Domäne eine Gruppe „PGUser" angelegt und die User dieser zugeordnet. Diese Gruppe wurde dann für alle freigegeben n Order zugelassen. Es wurde außerdem automatisch in regelmäßigen Abständen ein Backup von allen relevanten Daten (Benutzerprofile, Projektdateien, CVS-Repository, Datenbanken usw.) auf eine separate Festplatte durchgeführt, damit es zu keinem Datenverlust kommen konnte.
Für nahezu alle anderen Dienste konnten sich die PG-Mitglieder ebenfalls mit ihren Usern aus der Domäne anmelden, hierunter fallen unter anderen:

- CVS
- WWW
- FTP
- News

Für die Dienste Application Server (Weblogic), Oracle Datenbank und Netscape Directory Server wurden separate User eingerichtet, da diese Dienste auch standalone auf anderen Server laufen könnten (beispielsweise in einer verteilten Umgebung) und sie somit keine Abhängigkeit vom PDC haben.

10.2 Server

Wichtigstes Element auf dem Server ist der Bea Weblogic 7.0 als Application Server. Wir haben diesen Server gewählt, weil er bereits EJB 2.0 unterstützt, und somit die einfache Entwicklung von Beans mit Container Managed Persistence erlaubt. Dies befreit uns von dem Aufwand komplexe SQL-Entwicklung zu betreiben, um unsere Beans persistent zu machen. Ein weiteres Argument sind die hervorragenden Tools und Administrationswerkzeuge, sowie eine frei verfügbare Dokumentation im Internet [Bea03]. Von dieser Kombination versprachen wir uns eine deutliche Zeitersparnis gegenüber der Nutzung anderer Server, insbesondere gegenüber Open-Source Lösungen.
Die beiden anderen Elemente sind zum einen die relationale Datenbank von Oracle, die unsere EntityBeans speichern wird und der Netscape Directory Server, der zur Authentifizierung und Profilverwaltung dient.
Der Bea Weblogic wurde auf dem PG-Server zwei mal installiert: Eine erste Instanz stellt den Routine-Server dar, hier läuft die jeweils aktuelle stabile Version des Projektes. Die zweite Instanz wird nicht auf dem PG-Server gestartet, sondern kann von den Arbeitsstationen aus gestartet werden (sog. „privater Server"). Auf dem PG-Server befinden sich also nur die Dateien des Weblogics, die über eine Laufwerksfreigabe von den Clients aus erreichbar sind.
Die zweite Instanz wurde aber im Gegensatz zur ersten Instanz speziell so modifiziert, dass sie über das Netzwerk ausführbar ist und das sie die Projektdaten nicht aus dem Verzeichnis des Programms sondern aus dem Profil des aktuellen Users startet. Jeder User hat dazu die Option, sich den aktuellen (im Live-Server deployten) Projektstand in seinen privaten Server zu kopieren, diesen dann zu starten und dann ausschließlich mittels seines privaten Servers zu entwickeln.

Um die Ergebnisse der Projektgruppe auch anderen Personen zugänglich zu machen, wurde von der PG eine Domain registriert (mcu.be), über die die Webseiten der PG abgerufen werden können. Damit die Teilnehmer auch von außerhalb Zugriff auf den Application Server zu haben, ist er auch von außen über eine geschützte Verbindung nutzbar. Dies ermöglicht mit dem Webinterface des Servers auch die Entwicklung von zu Hause. Um die Kommunikation innerhalb der Gruppe zu vereinfachen, kommt hierzu noch ein Mailverteiler, der dafür sorgt, dass einzelne PG-Mitglieder bequem alle anderen informieren können. Darüber hinaus gibt es auch einen Mailverteiler, um mit der anderen PG zu kommunizieren. Dazu kommen dann noch ein News-Server, ein FTP-Server und ein privater Bereich auf der Homepage, der zum Beispiel wichtige Literatur und Nachschlagewerke in elektronischer Form bietet.

Außerdem wurde das Tool „CruiseControl“ eingesetzt, näheres dazu findet sich später in einem eigenen Abschnitt.

10.3 Clients

Auch auf den Clients wurden Tools installiert, die zum Arbeiten unverzichtbar sind. Dazu gehören zum Beispiel ein Webbrowser und ein Handyemulator von Nokia, der es ermöglicht, xHTML und WML Seiten darzustellen. Ein aktueller Webbrowser wird außerdem vorausgesetzt, um das Webinterface des Bea Weblogic zu benutzen. Um die in letzter Zeit vor allem zu Dokumentationszwecken immer häufiger auftauchenden PDF-Dokumente anzeigen zu können, wird der Acrobat Reader der Firma Adobe bereitgestellt. Um schließlich selber dokumentieren zu können, wird zumindest ein Textverarbeitungssystem benötigt. Wir haben uns hier für LaTeXentschieden; es bietet uns eine komfortable Möglichkeit, unsere Dokumentation in professionellem Layout zu erstellen. Außerdem bietet es noch die Möglichkeit, aus unseren Dokumenten PDF-Dokumente zu erzeugen. Um mit komprimierten Daten umgehen zu können, sind außerdem noch diverse (De-) Komprimierungsprogramme installiert worden. Zur Überprüfung der vom Application Server gespeicherten Daten wurde außerdem das Datenbanktool Free Toad installiert, das direkten Zugriff auf unsere Oracle Datenbank bietet. Die Administration des LDAP-Servers kann mit dem Java basierten Netscape Tool vorgenommen werden.

Um die Wartung der Rechner zu vereinfachen, wurde für jeden der beiden Typen von Clientrechnern ein komplettes Image erzeugt, das dann nur noch in die anderen Rechner kopiert werden musste. Um die Entwicklung und das Testen insbesondere der EJBs zu vereinfachen, wurde außerdem ein Skript geschrieben, das eine Kopie des aktuellen Bea Weblogic-Servers auch auf dem Client Rechner starten kann. Lokale Änderungen zum Testen einer Komponente wirkten sich so nicht auf andere Entwickler aus, die zur selben Zeit diese Komponente nutzen möchten. Eine funktionierende Variante aller Komponenten ist somit jederzeit am zentralen Server nutzbar.

10.4 Entwicklungswerkzeuge

Für die Entwicklung werden verschiedene Werkzeuge benötigt, die nachfolgend kurz vorgestellt werden sollen:

10.4.1 Sun JDK 1.3.1 und J2EE 1.3

Die Projektgruppe hat sich bei der Wahl der vorrangig zu benutzenden Programmiersprache für Java entschieden. Diese Sprache erlaubt volle objektorientierte Programmierung, ist plattformunabhängig, kostenlos und lizenzgebührenfrei. Sie ist außerdem hervorragend dazu geeignet, internetbasierte Anwendungen zu konzipieren und erlaubt eine direkte Anbindung an Webbrowser. Die Enterprise Edition bietet mit dem Konzept der Enterprise Java Beans ein ideales Komponentenmodell, um die Business Logik unserer Anwendung zu realisieren. Und auch für die Darstellungsschicht bietet sich Java mit dem Konzept der Servlets und JSPs an. Es ermöglicht die Umsetzung eines MVC-Konzeptes und eine einfache Anbindung von unterschiedlichen Clients (z.B. HTML, xHTML und WML). Wir haben uns bewusst für die Version 1.3.1 entschieden, auch wenn die Version 1.4 bereits erschienen ist. Wir müssen keine neuen Features der Version 1.4 zwingend nutzen, und uns sind gravierende Probleme mit dem neuen JDK aufgefallen. Interne Änderungen führen zu vielen Inkompatibilitäten mit den Entwicklungsumgebungen und dem Server. Somit haben wir uns auf 1.3.1 verständigt, um unnötiges Fehlersuchen zu vermeiden.

10.4.2 CVS / WinCVS

Um während der Entwicklung eine Versionskontrolle durchführen zu können, benötigt die Gruppe ein Versionsverwaltungssystem. CVS ist frei erhältlich und unterstützt Client/Server-basierte Versionskontrolle. Es gibt eine Version für Windows NT, so dass wir dieses System auf unserem zentralen Server einsetzen können. Clients sind für verschiedene Betriebssysteme vorhanden (z.B. WinCVS für Windows).

10.4.3 Together 6.0

Um den objektorientierten Entwurf unseres Projekts zu unterstützen, kam für uns nur die objektorientierte Entwurfsmethode auf Basis von UML infrage. Together/J erwies sich hier als perfektes Tool. In den neuen Versionen wurde die Entwicklung von EJBs immer stärker unterstützt. Daher ist es neben dem Entwurf auch unsere bevorzugte Entwicklungsumgebung für die Enterprise Java Beans. Aus den Komponentenmodellierungen haben wir direkt die Java-Klassen erzeugt und weiterbearbeitet. Insbesondere die Hilfe bei der Erstellung der XML-Deskriptoren erwies sich als sehr hilfreich.

10.4.4 Bea Weblogic Console/Builder 7.0

Da Together in der von uns genutzten Version 6.0 leider nicht die zusätzlichen, herstellerabhängigen XML-Deskriptoren für den Bea Weblogic 7.0 erzeugen kann, sind wir zusätzlich auf das Deploytool von Bea angewiesen, um die letzten Einstellungen an den Beans vorzunehmen.

10.4.5 Buildprozess mit Ant und CruiseControl

Durch den im Vergleich zu normalen Java-Komponenten recht komplizierten Vorgang, eine EJB-Komponente zu kompilieren und im Server zu deployen, um sie schließlich benutzen und testen zu können, entstand bei uns der Wunsch, diesen Vorgang zu vereinfachen und zu automatisieren. Der Vorgang, Enterprise Java Beans zu erstellen und zu deployen, teilt sich in folgende Schritte:

- Remote-Interface, Home-Interface und Bean-Implementierungsklassen erzeugen und konsistent halten
- Deployment-Deskriptoren erzeugen
- Beans zusammen mit den Deployment-Deskriptoren in eine jar-Datei packen
- JNDI-Namen und Classpath so konfigurieren, dass sich alle Beans untereinander ansprechen und benutzen können und auch die Webanwendungsschicht Zugriff auf die Beans hat
- Alle Beans und die Webanwendung gemeinsam in den Server deployen

Da wir für den Entwurf unsere Komponenten mit dem Softwaretool Together erstellt hatten, begannen wir mit diesem Tool auch die Erstellung und Implementierung der Beans. Das Programm bot uns gute Hilfestellung und Validierungstools zum Erstellen der Interfaces und der Deskriptoren. Uns fehlte also noch eine Automatisierung der Schritte 3-5.
Für die restlichen Schritte fanden wir als Lösung eine Kombination aus den beiden Open Source Tools Ant und CruiseControl.
Ant [TB02] ist ein Projekt der Apache Foundation, die auch den Apache Webserver entwickelt und pflegt. Weitere Informationen und Downloads zu diesem Tool findet man auf der Projekthomepage unter http://ant.apache.org
Ant ist ein Java-basiertes Build Tool, das mit dem bekannten Unix „make“ zu vergleichen ist. Es basiert jedoch vollständig auf Java und lässt sich daher problemlos auf jeder Plattform einsetzen, für die es eine Java Virtual Machine gibt. Darüber hinaus gibt es viele so genannte Tasks, um verschiedene Aufgaben zu erfüllen. Die Aufgaben lassen sich durch verschiedene Targets gruppieren. Ein Target kann beliebig viele Einzelaufgaben zusammenfassen, und zusätzlich kann die Ausführung von anderen Targets abhängen, oder ein Target kann auch andere Targets auslösen. Zusätzlich kann man durch Konfigurationsdateien leicht Angaben von Pfaden etc. anpassen. Dies war besonders hilfreich, da wir so leicht die unterschiedlichen Pfade für den privaten Testserver und den Liveserver ändern konnten. Somit konnten wir leicht Targets für das Kompilieren der Einzelkomponenten oder auch der gesamten Applikation erstellen und das Deployen in den Server automatisieren. Dieses Tool würde also bei der Nutzung am Arbeitsplatz viele Schritte vereinfachen. Da wir jedoch in mehreren Teams arbeiteten, wollten wir eine zentrale Kontrolle des Arbeitsfortschritts und der Arbeitsfähigkeit des Systems. Dabei stießen wir auf das Sourceforge Projekt CruiseControl. Es unterstützt die Idee der „Continous Integration“ d.h. während des gesamten Entwicklungsprozesses werden automatisch und in regelmäßigen Zeitintervallen alle Teile des Projekts zusammengefügt und getestet [FF03]. Die automatisierten Tests werden mit Hilfe des Testing Frameworks JUnit durchgeführt, das das Prinzip des Unit Testings unterstützt [LF02]. Die meisten dieser Tools sind zur Unterstützung des „eXtreme programming“ Entwicklungsmodells entstanden [Bec99]. Wir haben jedoch nur die Tools genutzt, ohne „eXtreme programming“ als Entwicklungsmodell einzusetzen.
Mit diesem Tool konnten wir nun die Ergebnisse des Vorgangs automatisch auf einer Webseite veröffentlichen. Falls ein Test fehlschlägt, geht sofort eine Benachrichtigungsmail an die Teammitglieder. Somit werden Fehler schnell entdeckt, und die Integration der Komponenten wird enorm erleichtert.
Um sich die Nutzung des Tools vorstellen zu können, findet sich in Abbildung 10.1 ein Screenshot von unserer Webseite.

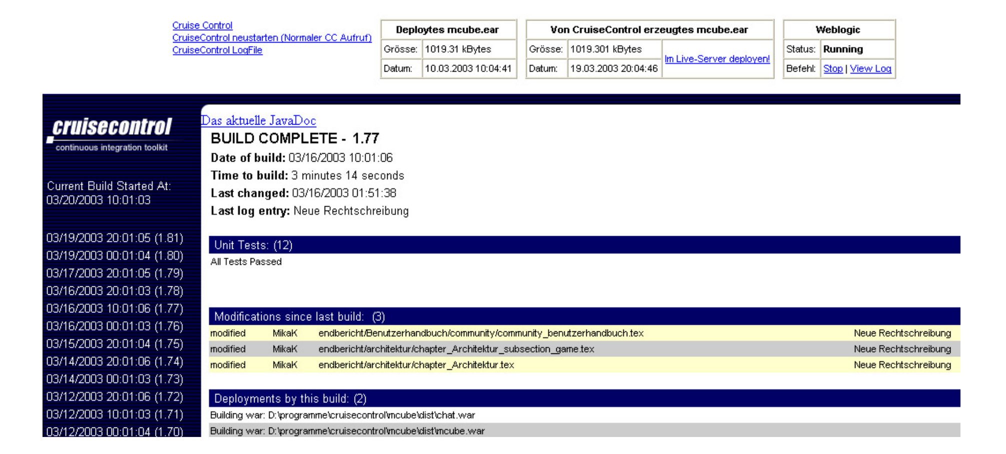


Abbildung 10.1: CruiseControl Webseite

In der oberen Leiste finden sich Funktionen zum Neustarten des Bea Weblogic und CruiseControl, außerdem ist es möglich Informationen über die das Logfile und die Größe der .ear Dateien zu sehen und gegebenenfalls eine neue Version zu deployen.
Am linken Rand findet man eine Auflistung der vorangegangenen Builds. Bei einem Klick auf eine bestimmte Version erscheinen im zentralen Frame Informationen zu diesem Durchlauf.
Ganz oben kann man zunächst das aktuelle Java-Doc einsehen. Darunter finden sich dann allgemeine Informationen, wie Uhrzeit, Dauer und Erfolg, zu dem Durchlauf. Darunter findet man dann die Details: Erfolgs- oder Fehlermeldungen der Unit-Tests, eine Auflistung der CVS-Änderungen seit dem letzten Build und eine Liste der geänderten Dateien.
Die von uns angepasste CruiseControl Intranet Seite bietet also einen zentralen Anlaufpunkt für den Entwickler. Hier kann man den Fortschritt des Projekts überprüfen und auch die wichtigsten administrativen Dinge für das Deployment erledigen. Die neuen Sourcen müssen nur ins CVS eingescheckt werden, und beim nächsten Durchlauf von CruiseControl werden die Komponenten automatisch aktualisiert, getestet und gegebenenfalls auch deployed.
Das Ant-Buildfile alleine kann ohne CruiseControl auch bei der lokalen Entwicklung auf dem privaten Testserver genutzt werden. So kann der Komfort des automatischen Kompilierens und Deployens auch lokal genutzt werden.

10.4.6 Probleme

Bei der Umsetzung dieses Konzeptes hatten wir das Problem, dass die Komponenten die Remote-Interfaces der anderen Komponenten nicht finden konnten, da sie nicht im Classpath waren. Dies führte dazu, dass wir zu Beginn die Remote-Interfaces in alle jars hineinkopiert haben. Dies war jedoch keine zufrieden stellende Lösung, da sie durch die mehrfach vorhandenen Interfaces leicht zu Inkonsistenzen und Classpath Problemen führen könnte. Also entschlossen wir uns dazu alle benötigten Komponenten jars und die Webapplication wars in ein so genanntes Enterprise Application Archiv (EAR) zu packen, so kann man in der Manifest Datei des jars den Classpath angeben und auf diesem Wege die benötigten jars dem Classpath der Komponente hinzufügen.

Beim Deployen der Komponenten stellten wir jedoch fest, dass die Komponenten zur Erzeugung der Stubs und Skeleton Klassen neu kompiliert werden. Dabei werden die Classpath Informationen aus dem Manifest leider ignoriert, und das Kompilieren schlägt fehl. Daher haben wir ein weiteres Ant-Target erzeugt, das die Klassen vor dem Deployen erzeugt. Dies gelang durch das Benutzten eines von Weblogic zur Verfügung gestellten externen Compilers. Hier konnten wir den Classpath explizit angeben, und die Probleme waren gelöst, denn nach dem Vorkompilieren deployten die Komponenten, und die Classpath-Informationen wurden genutzt.

Kapitel 11

Implementierung

Wie bereits beschrieben haben wir ausgehend von dem Architekturdokument zunächst einen Durchstichprototypen entwickelt und dann die weitere Implementierung in mehreren Inkrementen daran angeschlossen.
Für die Entwicklung haben wir uns in Kleingruppen aufgeteilt, die jeweils für einzelne Komponenten verantwortlich waren.
Während der Prototyp-Phase hat das Architekturdokument fortwährend Änderungen erfahren, die allerdings nur auf der Ebene der einzelnen verwendeten Funktionen stattgefunden haben. Dafür - wie auch für die Entwicklung der weiteren Inkremente - war die starke Kommunikation unter allen Kleingruppen äußerst hilfreich. Wir beenden die Projektgruppe mit dem positiven Ergebnis, die selbst gestellten Anforderungen weitestgehend erfüllt zu haben.
Im Folgenden wird die Implementierung der einzelnen Komponenten näher erläutert.

11.1 Usermanagement-Komponente

11.1.1 Beschreibung

Die Usermanagement Komponente ist eine Komponente der core-Schicht und setzt auf keiner Funktionalität der anderen Komponenten auf. Abhängigkeiten zu anderen Komponenten entstehen also indirekt nur durch die Benutzung der Usermanagementfunktionalitäten in anderen Komponenten.

Das Usermanagement bietet Funktionen zum Anlegen, Verwalten und Löschen von Userdaten. Dazu zählen insbesondere das Verwalten von Logindaten und Sicherheitsüberprüfungen sowie eine Speicherung der Datenprofile für die unterschiedlichen Anwendungen.

Es gibt ein festes Datenprofil für alle üblichen Daten wie Name, Vorname, Handynummer etc. Diese Daten werden im Community-Profil gespeichert. Für andere Komponenten kann jeweils ein eigenes Datenprofil definiert werden. In unserer Anwendung wurden Datenprofile für Game und Dating angelegt. Diese Komponente dient als Basis für die userbasierte Datenhaltung in allen Komponenten und arbeitet eng mit der Groupmanagement Komponente zusammen, mit deren Hilfe man die User auch in Gruppen anordnen und verwalten kann.

11.1.2 Realisierung

Die Usermanagement Komponente wurde als stateless SessionBean implementiert. Da die Komponente keine Funktionalität anderer Komponenten benutzt, müssen auch keine Verbindungen zu anderen Komponenten aufgebaut werden. Da wir allerdings eine LDAP-Datenbank benutzen, muss in der ejbCreate-Methode eine Verbindung aufgebaut und als Referenz gehalten werden. Alle Parameter, um die LDAP-Datenbank anzusprechen (Login, Passwort, ConnectionFactory etc.), werden aus der Constants Klasse ausgelesen, damit die Daten an einem zentralen Ort gepflegt werden können.
In der Planungsphase waren wir noch davon ausgegangen, die Funktionalität mithilfe von EntityBeans zu implementieren. Wir haben uns nach Abwägung der Alternativen dafür entschieden, statt dessen eine LDAP-Datenbank einzubinden. Für diese Entscheidung sprachen im Wesentlichen zwei Faktoren. Zum einen bietet jeder LDAP-Server Funktionalitäten, um sowohl die Passwortverwaltung mit automatischer Verschlüsselung und Gültigkeitszeiträumen von Passwörtern zu unterstützen als auch die Suche in den Daten mit Stringoperationen zu vereinfachen. Zum anderen sprach der Anspruch auf eine leichte Erweiterbarkeit und Zusammenarbeit mit anderen Anbietern, die andere Funktionalitäten beisteuern, für LDAP. Da LDAP schon seit langer Zeit ein Standard ist, wird er in vielen Bereichen zu Verwaltung der Userdaten eingesetzt. Anbieter von Internet-Community-Diensten oder eMail-Provider könnten ihren Usern so leicht den Zugang zu unseren Systemen erleichtern, da einfach die schon vorhandenen User- und Logindaten benutzt werden können und nur die neuen Daten zusätzlich gespeichert werden müssen.
Alle Methoden gibt es in zwei Ausführungen. Die eine führt jede Aktion ohne jede Überprüfung aus und ist für die Benutzung zu Administrationszwecken gedacht. Die andere Methode enthält als zusätzlichen Parameter jeweils den Namen des Users, der die Informationen abrufen möchte. So wird sichergestellt, dass ein User alle Daten seines Profils ansehen und ändern kann, ein anderer User jedoch nur die Daten erhalten kann, die er sehen darf.
Um den Datenverkehr zu vermindern, wurde das Value-Object Pattern eingesetzt. Will man viele Daten eines Users erhalten - z.B. für die Anzeige aller Daten in einer Übersicht - kann man ein User Objekt anfordern, das alle Daten eines Users enthält. So ist nur eine Kommunikation mit dem Server nötig, um alle Daten zu erhalten.
In alle Methoden der Komponente wurde das Logging integriert, so dass die einzelnen Arbeitsschritte in der Logdatei des Applicationservers nachvollzogen werden können. Hierbei wurden einzelne Stufen des Loggings von Debug über Info und Error bis Fatal verwendet, um den Informationsgehalt richtig einordnen zu können. Außerdem verfügt die Komponente über ein komplettes Exceptionhandling.
Insgesamt konnten also mithilfe der Anbindung an LDAP alle Anforderungen erfüllt werden.

11.1.3 Probleme

Bei der Implementierung der Usermanagement Komponente traten keine größeren Probleme auf. Zunächst mussten wir uns natürlich in die LDAP Thematik einlesen, den Server einrichten und geeignete Java-Klassen finden, um den Server anzusprechen, da es mehrere leicht unterschiedliche Implementierungen gibt (Novell, Netscape und Sun).
Nachdem wir den Service eingerichtet hatten und ansprechen konnten, mussten wir uns eine Struktur zur Speicherung der Daten überlegen, die sich einerseits schnell durch-

suchen lässt und andererseits für Profile neuer Anwendungen erweiterbar ist. Dabei stießen wir auf das Problem, dass sich das Schema der LDAP-Datenbank nicht über die Java-Ansprache ändern lässt. So ist es leider nötig, die Anpassungen für ein neues Profil für eine zusätzliche Anwendung mithilfe der LDAP-Server spezifischen Administrationskonsole vornehmen zu lassen. Da dies nicht so häufig vorkommt, halten wir es allerdings für eine akzeptable Einschränkung. Zumal für eine neue Komponente meistens auch andere Dinge durch den Administrator geändert werden müssen (neue JMS-Queues, neue Datenbanktabellen etc.).
Eine zusätzliche Herausforderung war das Mapping von gängigen Java-Datentypen auf LDAP-Attributtypen. Denn Java-Objekte lassen sich zwar auch direkt im LDAP-System speichern, aber dann werden sie als Binärstream gespeichert und können nicht sinnvoll durchsucht werden. Also schrieben wir Methoden, die bei der Speicherung Java-Objekte auf passende LDAP-Attributtypen mappen und beim Auslesen aus den LDAP-Informationen wieder ein Java-Objekt erzeugen.
Die restlichen Anforderungen konnten ohne größere Probleme erfüllt werden.

11.1.4 Verbesserungs- und Erweiterungsmöglichkeiten

Die Usermanagement Komponente hat alle Anforderungen aus der Planungsphase erfüllt, dennoch haben wir einige Ideen angesichts der knappen Zeit nicht mehr umsetzen können. Die erste Verbesserungsmöglicheit wäre ein ausgefeilteres Berechtigungssystem auf Rollenbasis, das sich Grundfunktionalitäten der Java Security Mechanismen bedient. Die zweite Verbesserungsmöglichkeit wäre eine Auslagerung der LDAP-Konfiguration in eine XML-Datei. So könnte die Komponente auch ohne Neukompilierung der Constants Klasse für einen neuen Server eingerichtet werden. Die Anpassungen wären damit auch für den Bean-Deployer möglich.

11.2 Groupmanagement-Komponente

11.2.1 Beschreibung

Die Groupmanagement-Komponente ist in der core-Schicht angesiedelt und dient zur Gruppenverwaltung. Es können neue Gruppen angelegt und wieder gelöscht werden. Einer Gruppe können User hinzugefügt und entfernt werden. Dabei besteht eine Gruppe nur aus Usern. Es gibt also keine Untergruppen. Man kann sich sowohl alle User einer Gruppe als auch alle Gruppen, zu denen ein User gehört, anzeigen lassen. Außerdem kann man überprüfen, ob ein User zu einer bestimmten Gruppe gehört.
Die Gruppenverwaltung wird beispielsweise von der Community benutzt, um die Buddylisten der User zu verwalten.

11.2.2 Realisierung

Die Funktionalität dieser Komponente wird durch ein stateless SessionBean bereitgestellt. Außerdem hat die Komponente noch zwei EntityBeans, um die Gruppen und die Zugehörigkeit der User zu diesen Gruppen persistent in der Datenbank speichern zu können. Die Verbindung zu diesen beiden benötigten Beans wird bei der Erzeugung des Groupmanagement Beans hergestellt.
Das EntityBean Group repräsentiert die Tabelle Groups, die als einziges Attribut den Gruppennamen besitzt, der zugleich auch Primary Key ist. Hier werden also die Na-

men aller Gruppen gespeichert. Nur mit den hier registrierten Gruppen kann operiert werden.
Das zweite EntityBean GroupUsers repräsentiert die Tabelle GroupUsers. Als Attribute beinhaltet sie den Gruppen- und Usernamen. Zusammen ergeben sie den Primary Key. Ein Datensatz speichert also die Zugehörigkeit eines Users zu einer Gruppe.
Damit die Datenintegrität gesichert ist, handelt es sich bei dem Gruppennamen um einen Fremdschlüssel von der Tabelle Groups. Somit können User nur zu Gruppen zugeordnet werden, die wirklich existieren.
Hier folgt nun eine detaillierte Beschreibung der Funktionalität:
Um eine neue Gruppe anzulegen, ruft man die Methode createGroup(Gruppennamen) auf. Falls der Gruppenname schon existiert, wird eine DuplicateGroupException geworfen. Andernfalls kann man nun dieser Gruppe User zuordnen.
Wenn eine Gruppe nicht mehr benötigt wird, sollte diese mithilfe der Methode deleteGroup gelöscht werden. Hierbei werden auch alle Zuordnungen der User zu dieser Gruppe gelöscht. Dank der Fremdschlüsselvererbung in der Datenbank und Einschalten der Funktion Cascade Delete wird dies automatisch von der Datenbank übernommen. Neben der Integrität der Daten birgt dieses Vorgehen außerdem einen Performanz Vorteil, da die entsprechenden Datensätze nicht auf der Entity Bean Ebene gelöscht werden.
Einen User fügt man mit der Methode addUserToGroup(group, user) hinzu. Die Methode erzeugt dazu ein neues GroupUsers EntityBean mit der Gruppe und dem User. Mit deleteUserFromGroup(group, user) kann ein User wieder aus der Gruppe entfernt werden. Das entsprechende GroupUsers EntityBean wird gelöscht.
Mit getUsersFromGroup(group) erhält man eine Liste mit allen Usernamen, die der Gruppe angehören. Um diese Funktion zu realisieren, wurde eine spezielle FindBy-Methode mithilfe der in der EJB Spezifikation 2.0 definierten Query Language geschrieben, die alle Datensätze in der Tabelle GroupUsers findet, in denen der Gruppenname vorkommt. Wenn man eine Liste mit allen Gruppen, denen ein User angehört, haben möchte, so muss man die Methode getGroupsFromUser(user) aufrufen. Diese Methode arbeitet ähnlich wie getUsersFromGroup, nur dass hier eine andere FindBy-Methode benutzt wird, die nach Datensätze mit dem Usernamen sucht.
Die Methode isMemberOfGroup(group, user) gibt an, ob ein User Mitglied einer Gruppe ist oder nicht.
Zum Exceptionhandling der Komponente ist noch Folgendes zu sagen: Falls einer Methode als Parameter null statt einem Namen übergeben wird, so wird eine ParameterIsNullException geworfen. Außerdem überprüft jede Methode, ob wirklich eine Gruppe mit dem übergebenem Namen existiert. Falls nicht, wird eine GroupNotFoundException geworfen. Damit ist sichergestellt, dass man einen User nicht einer Gruppe zuordnen kann, die überhaupt nicht existiert. Dieses Vorgehen ist notwendig, um die Datenintegrität zu gewährleisten.
Entsprechend der Vorgehensweise im Usermanagement wurde gleichermaßen das Logging integriert.

11.2.3 Probleme

Bei der Implementierung dieser Komponente traten keine nennenswerten Probleme auf.
Da diese Komponente ganz unten auf der core-Schicht angesiedelt ist, ist sie von keiner anderen Komponente abhängig.

Ein aufgetretenes Problem war, dass man einer Gruppe einen User mehrfach zuweisen konnte. Dies wurde dadurch gelöst, dass der Primary Key von einer ID durch einen zusammengesetzten Primary Key ersetzt wurde, bestehend aus den Attributen Gruppennamen und Usernamen. Mehrfachzuweisungen sind somit nicht mehr möglich.

11.2.4 Verbesserungs- und Erweiterungsmöglichkeiten

Eine mögliche Erweiterung wäre, dass man Untergruppen zulassen könnte.

11.3 Locationing-Komponente

11.3.1 Beschreibung

Die Locationing Komponente ist eine grundlegende Komponente der core-Schicht, da sie eine Basisfunktionalität bereitstellt, die von anderen Komponente wie Routing und FriendsAround genutzt wird. Diese Basisfunktionalität ist die Lokalisierung von Personen und Objekten, ohne die beispielsweise ein FriendsAround Service gar nicht möglich wäre.

11.3.2 Realisierung

Die zentrale Aufgabe der Positionsbestimmung übernimmt das LocationingBean, das als stateless SessionBean implementiert wurde. Es nutzt zur Erfüllung der an das Bean gestellten Anforderungen drei EntityBeans (LocationingCache, ObjectPosition und PLZ) und die Klasse Position.
Ein Objekt vom Typ Position beschreibt den Aufenthaltsort einer Person, den Standpunkt eines Objektes oder den Ort, der durch eine Postleitzahl beschrieben wird, mittels der zwei Attribute für Längen- und Breitengrad der Koordinaten und einem weiteren Attribut, das die mögliche Abweichung von den Koordinaten in Metern enthält. Objekte vom Typ Position werden von den Lokalisierungsmethoden returniert, während die eigentlichen Koordinaten des Aufenthaltsortes eines Users, seine ID, seine Handynummer und der Zeitpunkt seiner letzten Ortung über das EntityBean LocationingCache gespeichert werden. Die komplette Beschreibung eines Objektes inklusive seiner ID, seiner Postadresse, einer Beschreibung und vor allem der Koordinaten seines Standpunktes wird mit dem EntityBean ObjectPosition persistent gemacht. Das EntityBean PLZ speichert zu allen Postleitzahlen in Deutschland die entsprechenden Längen- und Breitengrade.
In der Methode ejbCreate des LocationingBeans wird daher zunächst eine Verbindung zu den EntityBeans ObjectPosition, LocationingCache und PLZ etabliert. Außerdem wird noch eine Verbindung zum Usermanagement hergestellt. Dies geschieht aus Gründen der Sicherheit, da ein User die Möglichkeit hat, durch Setzen des Attributs positioningallowed in seinem Community Profil seine Ortung zu genehmigen oder zu untersagen, falls er nicht lokalisiert werden möchte. Die Methode isPositioningAllowed fragt dieses Attribut über eine Methode des Usermanagements ab.
Die beiden zentralen Methoden des LocationingBeans sind jedoch getUserPosition und getObjectPosition. Die Methode getUserPosition erhält als Parameter die ID des zu lokalisierenden Users, sowie einen Wert für den QualityofService. Zunächst wird mithilfe der gerade beschriebenen Methode isPositioningAllowed geprüft, ob der User seiner

Ortung zustimmt. Ist dies nicht der Fall, wird unmittelbar eine Exception vom Typ PositioningNotAllowedException geworfen, die extra für das LocationingBean erzeugt wurde. Andernfalls hängt das weitere Vorgehen davon ab, ob der User mit seinem Handy oder mit seinem Computer eingeloggt ist. Dazu wird beim Einloggen in der Community das Attribut logintype für den User gesetzt. Das hat den Vorteil, dass diese Methode selbstständig das richtige Verfahren zur Positionsbestimmung auswählen kann, und sich die aufrufenden Komponenten nicht darum kümmern müssen.

Beim Handy wird die Position des Handys bestimmt unter der Berücksichtigung des gewünschten QualityofService. Hat dieser den Wert 0, bedeutet dies, dass eine aktuelle Positionsbestimmung benötigt wird und diese somit neu ausgeführt werden muss. Die ermittelten Koordinaten für Längen- und Breitengrad werden dann in einem Objekt vom Typ Position returniert und gleichzeitig über das LocationingCacheBean zusammen mit dem Zeitpunkt der Lokalisierung gespeichert, wobei eventuell vorhandene, ältere Daten überschrieben werden. Ein QoS-Wert von 1 oder 2 bedeutet, dass die Positionsdaten bis zu zehn Minuten bzw. bis zu einer Stunde alt sein dürfen. Bei einem solchen Wert wird also zunächst geprüft, ob für den User überhaupt schon Positionsdaten im LocationingCache vorhanden sind und wenn dies der Fall ist, ob diese nicht älter sind als gefordert. Finden sich entsprechende Daten, werden diese in ein Objekt vom Typ Position geschrieben und returniert, andernfalls erfolgt eine Neubestimmung der Position des Users.

Ist der User aber mit seinem stationären Computer eingeloggt, wird die Position seines Wohnortes returniert. Dazu sind in einer extra Tabelle in der Datenbank zu allen Postleitzahlen in Deutschland die dazugehörigen Längen- und Breitengrade. Es wird also nun die Postleitzahl des Users mithilfe des Usermanagements ausgelesen und mittels des PLZ EntityBeans die entsprechenden Koordinaten aus der Datenbank ermittelt. Die Daten werden ebenfalls in ein Objekt vom Typ Position gekapselt.

Bei Objekten handelt es sich vornehmlich um mögliche Treffpunkte für User wie Diskotheken, Kinos oder Bars. Diese haben natürlich immer denselben Standpunkt, weshalb in der Methode getObjectPosition nur überprüft wird, ob zu der übergebenen ID eines Objektes Positionsdaten über das EntityBean ObjectPosition ermittelt werden können. Ist dies der Fall, werden diese Daten wiederum in ein Objekt vom Typ Position geschrieben und returniert, andernfalls wird eine Exception vom Typ `ObjectNotFoundException` geworfen, die eigens für diesen Fall erzeugt wurde.

Wiederum wurde wie oben bereits dargelegt das Logging eingebunden. Außerdem verfügt die Komponente über ein komplettes Exceptionhandling, wobei spezielle Fehlertypen (PositioningNotAllowedException und ObjectNotFoundException) für bestimmte Fälle erzeugt wurden.

Insgesamt erfüllt die Locationing Komponente alle Anforderungen im Kontext der Positionsbestimmung von Objekten und Personen und schafft somit die Grundlage für weiterführende Anwendungen wie das Routing oder FriendsAround. Generell ist diese Komponente eine der wichtigsten Komponenten, da ohne die Lokalisierung von Personen und Objekten die Erzeugung eines ortsabhängigen Service gar nicht möglich wäre.

11.3.3 Probleme

Bei der Implementierung der Locationing Komponente traten keine größeren Probleme auf. Es wurden jedoch im Laufe des Entwicklungsprozesses Anpassungen vorgenommen, die zunächst nicht vorgesehen waren (s. Verbesserungsmöglichkeiten).

11.3.4 Verbesserungs- und Erweiterungsmöglichkeiten

Besonderes Augenmerk bei der Implementierung wurde auf die Einführung eines QualityofService in Kombination mit der Speicherung von Positionsdaten über EntityBeans gelegt, was im ursprünglichen Entwurf der Komponente nicht vorgesehen war. Diese wichtige Erweiterung entstand im Gespräch mit einem Unternehmen aus der Telekommunikationsbranche, das auf die nicht zu vernachlässigenden Kosten einer Ortung verwies. Da nicht immer aktuelle Positionsdaten benötigt werden, können durch Abrufen von älteren Daten aus dem Cache Ortungen eingespart werden. Außerdem zahlt sich eine Ortung teilweise doppelt aus, weil sie zum einen aktuelle Daten liefert, wenn benötigt, und zum anderen noch die Daten für weniger zeitkritische Services liefert. Somit wird die Locationing Komponente modernsten Anforderungen realer Telekommunikationsumgebungen gerecht. Leider reichte die Partnerschaft mit der Telekommunikationsbranche nicht soweit, dass wir über Schnittstellen auf Testumgebungen und reale Ortungsverfahren zugreifen konnten. Aus diesem Grund mussten wir uns auf die zufällige Bestimmung von Koordinaten beschränken, die jedoch auf den Großraum Ruhrgebiet beschränkt wurden. Deshalb wäre die größte Erweiterungsmöglichkeit, eine Anbindung an eine reale Testumgebung in einem Folgeprojekt vorzunehmen. Zu diesem Zweck wurde bereits ein Flag definiert, welches extern über ein XML-File gesetzt werden kann und die Verwendung von Zufallswerten an- oder abschaltet.

11.4 Routing-Komponente

11.4.1 Beschreibung

Die Routing Komponente ist eine Komponente der core-Schicht und setzt direkt auf der Locationing Komponente auf. Sie bietet die Möglichkeit, basierend auf der Ortung von Usern oder den Koordinaten von Objekten bzw. Treffpunkten, relative Positionsdaten zu berechnen. Zu diesen relativen Positionsdaten zählen der Abstand eines Users zu einem anderen User oder Objekt, der Winkel zwischen ihnen, und die Himmelsrichtung, in die ein User gehen müsste, um einen anderen User oder ein Objekt zu erreichen. Darüber hinaus kann die Komponente die Koordinaten eines optimalen Treffpunkts für zwei User ermitteln oder ein Objekt als optimalen Treffpunkt für zwei User aus einer Liste vorgegebener Objekte bestimmen. Aufgrund dieser Funktionen bildet die Routing Komponente eine wichtige Basis für die FriendsAround Komponente.

11.4.2 Realisierung

Die Routing Komponente wurde als stateless SessionBean implementiert. Da die Komponente direkt auf der Locationing Komponente aufbaut und daher auf die Funktionen dieser Komponente zugreifen muss, wird in der Methode ejbCreate eine Verbindung zum LocationingBean hergestellt. Verbindungen zu anderen Beans werden zur Erfüllung der Anforderungen nicht benötigt. Die berechneten Daten werden in Objekten vom Typ RelativePosition gehalten. Zu diesen Daten zählen die IDs der beiden User, zwischen denen die relativen Positionsdaten (Distanz in Metern, Winkel und Himmelsrichtung) berechnet werden. Dabei werden die Positionsdaten immer ausgehend von User1 ermittelt, also definiert direction beispielsweise die Himmelsrichtung, in die User1 gehen müsste, um zu User2 zu gelangen. Des Weiteren verfügt ein Objekt vom Typ RelativePosition über die üblichen Set- und Get-Methoden.

Für die Berechnung der relativen Positionsdaten stehen jeweils drei verschiedene Methoden zur Verfügung. Eine berechnet die Distanz, den Winkel oder die Himmelsrichtung zwischen zwei Usern, deren IDs sowie eine Festlegung des QualityofService, übergeben werden, die zweite führt die Berechnung für zwei bereits bekannte Positionen, die als Parameter übergeben werden, durch, und die dritte Methode ermittelt die relativen Positionsdaten zwischen einem User, dessen ID übergeben wird, und einer ganzen Gruppe von Usern, deren IDs in Form einer LinkedList übergeben werden. Auch hier wird ein Wert für QualityofService mitgegeben, der definiert, wie aktuell die Positionsdaten der User sein müssen. Beispielsweise kann so definiert werden, ob die Ortung eines Users neu durchgeführt werden muss, oder ob auf etwas ältere Positionsdaten zurückgegriffen werden kann. Näheres dazu findet sich in der Beschreibung der Locationing Komponente, die diese Funktionalität implementiert und generell für die Ortung von Usern verantwortlich ist. Allen Methoden ist jedoch gemeinsam, dass sie jeweils die private Methode createRelativePosition aufrufen, die sämtliche relativen Positionsdaten ermittelt und in einem Objekt vom Typ RelativePosition speichert. Aus diesem Objekt wird dann mit der entsprechenden Get-Methode die gewünschte Größe (Distanz, Winkel oder Himmelsrichtung) ausgelesen und returniert. Herzstück der Routing Komponente sind also die privaten Methoden createRelativePosition, sowie getAngle und getDirection, die die Berechnungen für den Winkel und die Himmelsrichtung intern durchführen. Genauer gesagt gibt es zwei createRelativePosition Methoden, wobei die eine zwei UserIDs erwartet und die Position dieser User zunächst über die Locationing Komponente ermittelt, um dann die zweite Methode aufzurufen, die zwei Positionsobjekte erwartet und basierend auf diesen die Berechnungen durchführt.

Ein Positionsobjekt hält die Koordinaten einer Position, also Längen- und Breitengrad als Float mit einer Genauigkeit von fünf Nachkommastellen. Die Distanz wird dann nach folgender Formel berechnet:

$$d = \sqrt{((L2 - L1) \cdot \cos((B2 + B1)/2))^2 + (B2 - B1)^2 \cdot 60 \cdot 1852}$$

Dabei ist d die Distanz, L2 und L1 bezeichnen den Längengrad der Position von User2 und User1, und B2 und B1 entsprechen dem Breitengrad der Position von User2 und User1. Die Formel entspricht den gängigen Methoden zur Navigierung in der Seefahrt. Winkel und Himmelsrichtung werden dann in zwei separaten Methoden berechnet, wobei für die Winkelbestimmung die Längen- und Breitengrade der User verglichen werden. Auf Grundlage des berechneten Winkels wird User2 in Bezug auf User1 in einen Bereich eingeordnet und somit die Himmelsrichtung ermittelt. Die acht verschiedenen Himmelsrichtungen (Nord, Nordost, Ost, Südost, Süd, Südwest, West, Nordwest) erstrecken sich dabei jeweils über einen Winkel von 45°. Nach diesen Berechnungen werden die entsprechenden Werte im neu angelegten RelativePosition Objekt gesetzt.

Es gibt neben den bereits beschriebenen Methoden zum Auslesen von einzelnen Daten wie Distanz, Winkel und Himmelsrichtung auch zwei Methoden, die das komplette Objekt vom Typ RelativePosition returnieren. Auch hier gibt es die Möglichkeit, ein Objekt zwischen zwei Usern zu erhalten oder eine LinkedList von Objekten zwischen einem User und einer übergebenen Gruppe von Usern, wobei die Gruppe ebenfalls durch eine LinkedList beschrieben wird, die die UserIDs hält.

Neben der Berechnung der relativen Positionsdaten kann noch mithilfe der Methode getMeetingPlace ein optimaler Treffpunkt für zwei User, deren IDs übergeben werden, ermittelt werden. Hierzu ist die Methode getMeetingPlace überladen, wobei eine Methode die Koordinaten des günstigsten Treffpunkts in Form eines Objekts vom Typ

Position zurückliefert. Die Koordinaten werden dabei einfach durch Mittelwertbildung von Längengrad und Breitengrad der beiden Userpositionen bestimmt. Die zweite Methode erhält neben den beiden UserIDs noch eine LinkedList mit den IDs von Objekten. Diese Objekte können z.B. Bars, Discotheken oder Kinos sein. Aus dieser Liste wird dann ein optimaler Treffpunkt für die beiden User in Abhängigkeit ihres momentanen Aufenthaltsortes bestimmt. Dazu wird zunächst die Position des optimalen Treffpunkts mit der gerade beschriebenen Methode ermittelt und dann das Objekt herausgesucht, das zu dieser Position den geringsten Abstand hat. Die ID dieses Objekts wird abschließend returniert.

Logging und Exceptionhandling wurden wie bereits mehrfach dargelegt integriert.

Insgesamt können also mithilfe der privaten Berechnungsmethoden sämtliche relativen Positionsdaten für die verschiedensten Anwendungszwecke ermittelt werden. Dieser intelligente Aufbau von interner Berechnung, Speicherung der Daten in einem speziellen Objekt, Zugriff auf einzelne Daten oder das Gesamtobjekt sowie anspruchsvollerer Methoden wie der Ermittlung eines Objektes als optimalem Treffpunkt ermöglicht es, alle Anforderungen an die Routing Komponente zu erfüllen.

11.4.3 Probleme

Bei der Implementierung der Routing Komponente traten keine größeren Probleme auf. Zunächst musste recherchiert werden, um an die geeigneten Formeln zur Berechnung der relativen Positionsdaten zu gelangen. Als die interne Berechnung der Daten fertig gestellt war, konnten die Zugriffsmethoden leicht implementiert werden. Die Zugriffsmethoden wurden nachträglich überladen, um auch direkt die Distanz, den Winkel und die Himmelsrichtung zwischen zwei schon bekannten Positionen ermitteln zu können, was eine Anforderung der Dating Komponente darstellte. Dies ist ein Indiz für die problemlose Erweiterbarkeit der Komponente und ihr erfolgreiches Wachsen während des Entwicklungsprozesses, der Hand in Hand mit der Entwicklung der grundlegenden Locationing Komponente erfolgte, was von großem Vorteil war.

11.4.4 Verbesserungs- und Erweiterungsmöglichkeiten

Die Routing Komponente enthält alle grundlegenden Berechnungs- und Zugriffsmethoden, um anspruchsvollere Methoden erfolgreich darauf aufsetzen zu können. Ein Beispiel ist hierbei die Ermittlung eines Objektes als geeigneten Treffpunkt. Durch Kombination all dieser Methoden wäre es denkbar, ein zielgerichtetes Routing zu entwickeln, das nach Ermittlung von Position und Treffpunkt für zwei User diese sukzessive zu diesem Treffpunkt lotst. Dies könnte einfach durch ständig neue Positionsbestimmung der User und darauf basierender Neuberechnung der relativen Positionsdaten erfolgen. Es ist ferner denkbar, diese Daten dann mit Kartenmaterial zu kombinieren und auf einem mobilen Endgerät somit ein Navigationssystem zu schaffen, wie es aus Automobilen bereits bekannt ist. Dazu wären allerdings das Know-how und die Schnittstelle eines Telekommunikationspartners nötig gewesen, was uns leider nicht in vollem Umfang zur Verfügung stand. Somit mussten wir uns auf die Implementierung der Grundlagen beschränken, die jedoch die Basis für eine spätere Erweiterung, beispielsweise in einem Folgeprojekt einer anderen Projektgruppe, bilden.

11.5 Matching-Komponente

11.5.1 Beschreibung

Die Matching Komponente ist eine höherwertige Komponente der core-Schicht, da sie zur Erfüllung ihrer Aufgabe zwar nur auf die Usermanagement und die Routing Komponente zugreifen muss, aber dennoch einen Service realisiert, der speziell für die Dating Applikation sehr wichtig ist. Dieser Service umfasst die Suche nach Usern anhand von vorher definierten Kriterien, der so genannten MatchingAttribute. Es ist eine Suche nach allen passenden Usern, nach einer beschränkten Anzahl von entsprechenden Usern und nach einer beschränkten Anzahl von Usern, die einem bestimmten User ähnlich sind, möglich. Besonders wichtig ist in diesem Zusammenhang die Option einer ortsabhängigen Suche. Dabei kann ein Suchradius definiert werden, innerhalb dessen sich die durch das Matching gefundenen User befinden müssen, um in die Ergebnisliste der Suche aufgenommen zu werden. Der Radius wird ausgehend von der Position des suchenden Users angelegt, wobei die Position des Users entweder durch Ortung bei Verwendung eines Mobilfunkgerätes oder durch seine im Community Profil definierte Postleitzahl bestimmt wird.

11.5.2 Realisierung

Um diese Anforderungen zu erfüllen, wurde die Matching Komponente als stateless SessionBean implementiert, das sich zusätzlich der Funktionalität des Usermanagements und des Routings bedient, weshalb in der Methode ejbCreate eine Verbindung zum UsermanagmentBean und zum RoutingBean etabliert wird. Außerdem stehen die Klassen MatchingList und MatchingAttribute zur Verfügung.
Objekte vom Typ MatchingAttribute definieren die Kriterien, die einer Suche zu Grunde gelegt werden. Jeder User kann bestimmte Werte setzen, die in einer LDAP-Datenbank gespeichert sind. Er hat auf jeden Fall ein allgemeines Community-Profil mit Werten wie beispielsweise seiner ID, seinem Vor- und Nachnamen, seinem Geschlecht und seinem Geburtsdatum. Zusätzlich dazu kann er Profile für die Applikationen Game und Dating anlegen, wobei im Zusammenhang mit dem Matching vor allem das Dating-Profil eine wichtige Rolle spielt, da es Attribute wie Größe, Gewicht und Haarfarbe enthält. Die Objekte vom Typ MatchingAttribute ermöglichen nun, Kriterien zu definieren, die dann in der Suche mit den LDAP-Werten der User verglichen werden. Deshalb hält ein solches Objekt den LDAP-Namen eines Attributs, das Profil, in dem das Attribut vorkommt, und natürlich seinen Wert, wobei ein Start- und ein Endwert gesetzt werden können. Bei einem Attribut wie Haarfarbe spielt nur der Startwert eine Rolle, der dann z.B. den Wert „blond“ hat, während der Endwert null ist. Wenn aber nach einem Größenbereich von 170-175cm oder einem Alter von 18-23 Jahren gesucht werden soll, werden sowohl Start- als auch Endwert benötigt. Die Klasse MatchingAttribute verfügt natürlich auch über entsprechende get- und set-Methoden, um die Attribute zu setzen oder auszulesen.
Die Klasse MatchingList ist im Prinzip eine LinkedList, die Objekte vom Typ MatchingAttribute hält und die üblichen Methoden zum Verwalten und Iterieren der Liste bereitstellt. Es wurde hier eine spezialisierte Liste eingeführt, um lästige Castingoperationen zu verhindern und einen komfortablen Zugriff auf die einzelnen MatchingAttribute während einer Suche zu gewährleisten.
Für die Suche selbst stehen vier verschiedene Methoden zur Verfügung, wobei die Methode getAllSimilarUsers die grundlegendste ist. Sie erhält eine MatchingList mit Mat-

ching Attributen und konstruiert daraus einen Filterstring, der für die LDAP-Anfrage an das Usermanagement übergeben wird. Returniert wird schließlich eine LinkedList mit Usern, wobei es sich hier um eine Liste mit UserObjekten handelt und nicht um eine Liste mit den IDs der User wie bei den meisten Routing- oder FriendsAround-Methoden. Bei der Konstruktion des Filterstrings muss darauf geachtet werden, dass zwischen Attributen unterschieden wird, die nur einen Startwert haben (z.B. Haarfarbe), und solchen, die einen Suchbereich definieren (z.B. Größe). Außerdem muss das Attribut Alter besonders gehandhabt werden, da hier von einem Alter in ein entsprechendes Geburtsdatum umgerechnet werden muss. Grund dafür ist, dass in der LDAP-Datenbank das Geburtsdatum gespeichert ist und in der Suchabfrage der Dating Applikation ein Altersbereich definiert wird. Die Methode getAllSimilarUsers wurde für eine ortsabhängige Suche überladen. In der zweiten Version erhält sie als Übergabeparameter neben der MatchingList noch die ID des suchenden Users, da um dessen Position der Suchradius gezogen werden muss. Dieser Suchradius wird zunächst aus der Liste der Attribute herausgefiltert. Anschließend werden die passenden User mit der gerade beschriebenen ersten Version der Methode getAllSimilarUsers ermittelt. Mithilfe der Routing Komponente wird für jeden dieser User die Distanz zum suchenden User berechnet und überprüft, ob sich der User innerhalb des Radius befindet. Nur wenn dies der Fall ist, wird er der Ergebnisliste hinzugefügt.
Die Methode getSimilarUsers liefert im Prinzip die gleiche Funktionalität wie die erste Version der Methode getAllSimilarUsers, mit dem Unterschied, dass hier die Anzahl der Suchergebnisse eingeschränkt werden kann. Dazu werden zunächst mithilfe der Methode getAllSimilarUsers alle User bestimmt und dann die überflüssigen Ergebnisse aus der LinkedList entfernt.
Eine andere Funktionalität bietet die Methode getUserMatching, der die ID eines Users und ebenfalls eine Liste von Matching Attributen und eine Ergebnisbeschränkung übergeben wird. Sie sucht nach Usern, die mit dem User, dessen ID übergeben wurde, in den ebenfalls übergeben Kriterien übereinstimmt. Dazu werden die Start- und gegebenenfalls die Endwerte der Kriterien auf die Werte des Users gesetzt, bevor eine Suchanfrage durchgeführt wird. Dabei muss darauf geachtet werden, dass diese Attribute überhaupt für den User definiert sind. Diese Methode erlaubt es also beispielsweise, User mit ähnlichen Interessen oder Eigenschaften zu einem User zu finden, und ist daher eher für die Suche nach Freunden als für die Suche nach Flirtpartnern geeignet.
Wiederum wurden Logging und Exceptionhandling integriert.
Insgesamt erfüllt die Matching Komponente alle Anforderungen im Kontext der Suche von Usern. Dabei bietet sie ein breites Spektrum der Suche durch die verschiedenen Kombinationsmöglichkeiten von Matching Attributen und durch die Methoden mit unterschiedlicher Zielsetzung. Es ist ebenfalls jederzeit möglich, weitere LDAP-Attribute zu erzeugen oder komplett neue Profile für weitere Applikationen, die auf der mCube Plattform aufsetzen, zu definieren und dennoch weiterhin die Matching Komponente in vollem Umfang zu nutzen. Für den momentanen Stand des Projektes ist sie jedoch vor allem für die Dating Applikation unverzichtbar.

11.5.3 Probleme

Bei der Implementierung der Matching Komponente traten keine größeren Probleme auf. Lediglich die Konstruktion des Filterstrings in der Methode getAllSimilarUsers musste intensiv getestet werden, da hierbei zum einen auf verschiedenartige Attribute und zum anderen auf die korrekte Ansprache der Attribute in den einzelnen Profilen der LDAP-Datenbank geachtet werden musste.

11.5.4 Verbesserungs- und Erweiterungsmöglichkeiten

Die Matching Komponente erfüllt alle grundlegenden Anforderungen bezüglich der Suche von Usern, die für unser Projekt definiert wurden. Dennoch könnte sie in Folgeprojekten verfeinert werden, etwa hinsichtlich eines Rankings von Suchergebnissen. Dabei könnte man sich eine Sortierung der Ergebnisliste nach einem bestimmten Kriterium vorstellen. Ebenfalls könnte man eine Suchfunktion integrieren, die einzelne Attribute unterschiedlich stark gewichtet und dann einen Grad der Übereinstimmung mit der Suchanfrage berechnet. Die Vorbereitung dafür wurden bereits durch das Anlegen entsprechender Eigenschaften der Objekte vom Typ MatchingAttribute getroffen. Dies sind jedoch Feinheiten, die für die grundlegende Realisierung einer Matchingfunktion, wie sie für unser Projekt benötigt wurde, nicht nötig waren.

11.6 FriendsAround Komponente

11.6.1 Beschreibung

Die FriendsAround Komponente ist eine Komponente der core-Schicht, die direkt auf den weiteren core Komponenten Routing, Communication, Usermanagement und Groupmanagement aufsetzt. Der Grund dafür ist, dass sich die FriendsAround Komponente der Funktionalität der anderen Komponenten bedient, um die an sie gestellten Anforderungen zu erfüllen. Zu diesen Anforderungen zählt die komplette Bereitstellung eines Service, der in der mobilen Telekommunikation unter dem Namen FriendsAround bekannt ist. Hinter diesem Namen verbirgt sich die Möglichkeit eines Users, jederzeit herauszufinden, welche seiner Freunde in der Nähe sind, um eventuell mit diesen in Kontakt zu treten.

11.6.2 Realisierung

Um diesen Service realisieren zu können, wurde die FriendsAround Komponente als stateful SessionBean implementiert, so dass es eine Instanz dieser Komponente pro User gibt. Dazu wird bei der Erzeugung des Beans der Methode ejbCreate eine UserID übergeben, die lokal gespeichert wird. Alle weiteren Methodenaufrufe werden daraufhin ausgehend von diesem User betrachtet. Da für die Realisierung des FriendsAround Service die Funktionalität der core Komponenten Routing, Communication, Usermanagement und Groupmanagement benötigt wird, wird in der Methode ejbCreate eine Verbindung zu diesen Komponenten etabliert.
Bei der Beschreibung des FriendsAround Service wurde bereits erwähnt, dass für diesen Service die Möglichkeit herauszufinden, welche Freunde in der Nähe sind, essenziell ist. Dazu wird die Funktionalität der Routing Komponente genutzt. Das FriendsAroundBean stellt drei Methoden zur Verfügung, die die relativen Positionsdaten zwischen dem eigentlichen User, für den die Instanz des Beans angelegt wurde und dessen ID beim Erzeugen intern gespeichert wird, und einem Buddy, also einem seiner Freunde, dessen ID an die Methoden übergeben wird, ermitteln. Zu diesen relativen Positionsdaten zählen die Distanz und der Winkel zwischen dem User und seinem Buddy sowie die Himmelsrichtung, in die der User gehen müsste, um seinen Buddy aufzusuchen. Die Methoden getDistance, getAngle und getDirection liefern diese Daten, indem sie die entsprechenden Methoden des RoutingBeans aufrufen und dabei die IDs des Users und seines Buddys übergeben. Außerdem wird ein hoher Wert für den QualityOfService übergeben, wodurch das RoutingBean angewiesen wird, über

das LocationingBean auf ältere Positionsdaten aus dem Cache zurückzugreifen, falls diese vorhanden sind. Älter bedeutet dabei, dass die Positionsdaten bis zu 60 Minuten alt sein dürfen. Der Grund dafür liegt darin, dass die Ortung von Usern nicht zu vernachlässigende Kosten verursacht. Falls ein User nun herausfinden will, welche seiner beispielsweise 50 Freunde in der Nähe sind, wäre eine Neubestimmung der Position jedes Users einfach zu teuer und auch nicht nötig, da sich die Position eines Users innerhalb einer Stunde oft nicht deutlich verändert.
Es stellt sich nun noch die Frage, welche User überhaupt als Buddys des eigentlichen Nutzers der FriendsAround Komponente betrachtet werden. Die Antwort darauf lautet, dass der User seine Buddys in Listen oder Gruppen verwaltet und jederzeit neue User in diesen Gruppen aufnehmen kann, wodurch sie zu Buddys werden. Der Begriff Gruppe verdeutlicht dabei schon, dass hierzu die Funktionalität des GroupmanagementBeans genutzt wird. Zunächst muss eine leere Gruppe angelegt werden, wozu die Methode setFriendsAroundList bereitsteht. Diese erzeugt durch Aufruf der Methode createGroup des Groupmanagements eine leere Liste mit dem an die Methode übergebenen Namen. Außerdem wird über das UsermanagementBean das Attribut friendsaroundlist des Community-Profils für den User auf den Namen dieser Liste gesetzt. In diesem Attribut des Users steht also die Bezeichnung der momentan verwendeten FriendsAround Liste. Zusätzlich wird dieser Name noch lokal in der Variablen buddyList gespeichert, um nicht jedes Mal über das Usermanagement auf diesen Namen zugreifen zu müssen. Um die neue Liste mit Buddys zu füllen oder einzelne Buddys wieder zu entfernen, gibt es die beiden Methoden addUser und removeUser, denen jeweils die ID des Buddys übergeben wird. Die Methode addUser ruft eine entsprechende Methode des Groupmanagements auf, die den Buddy in die momentan aktuelle FriendsAround Liste einfügt. Außerdem wird der Buddy noch mit einer Nachricht darüber informiert, dass er in die Liste des Users aufgenommen wurde. Auch die Methode removeUser zum Entfernen eines Buddys aus der aktuellen Liste nutzt eine entsprechende Funktion des Groupmanagements. Abgerundet wird diese Gruppenverwaltung durch die Methode getMembers, die sämtliche Mitglieder der aktuellen Liste in Form einer LinkedList mit den IDs der Buddys ausgibt, wobei wiederum eine Funktion des GroupmanagementBeans aufgerufen wird.
Darüber hinaus besteht noch die Möglichkeit, einzelne Attribute aus dem Profil eines Buddys abzufragen und Nachrichten an Buddys zu senden und zu empfangen. Die Attributabfrage erfolgt mittels der Methode getAttribute, der das abzufragende Attribut und die ID des Buddys übergeben wird. Zur Erfüllung der Aufgabe wird eine Methode des Usermanagements verwendet. Das Senden und Empfangen von Nachrichten wird durch die Methoden sendMessage und getMessages geregelt, die diese Aufgabe durch eine Verbindung zum CommunicationBean, die ebenfalls in der Methode ejbCreate etabliert wird, erfüllen.
Das Logging mit Log4J wurde in sämtliche Methoden der Komponente eingebunden, um einzelne Arbeitsschritte in einer Logdatei, die sich auf dem Applicationserver befindet, zu sichern. Mithilfe dieser Datei kann dann die Arbeitsweise nachvollzogen werden. Für das Loggen existieren verschiedene Grade, mit denen eine Aussage in einen gewissen Kontext eingeordnet werden kann. Außerdem verfügt die Komponente über ein komplettes Exceptionhandling, das insbesondere mit der Groupmanagement Komponente koordiniert ist.
Insgesamt zeigt die Realisierung der FriendsAround Komponente, wie durch Bündelung der Funktionalität mehrerer core Komponenten, in diesem Fall Routing, Groupmanagement, Usermanagement und Communication, ein höherwertiger Service kreiert werden kann, der die an ihn gestellten Anforderungen durch das Zusammenspiel dieser

Komponenten erfüllt.

11.6.3 Probleme

Bei der Implementierung der FriendsAround Komponente traten keine größeren Probleme auf. Einzig der vielfältige Zugriff auf Funktionen verschiedener anderer core Komponenten gebot eine vorsichtige Implementierung, die sich strikt an den in der Architektur und der Anforderungsanalyse herausgearbeiteten Schnittstellen dieser Komponenten orientieren musste. Die erfolgreiche Entwicklung des FriendsAround Service ist somit ein Indiz dafür, dass in dieser Phase des Projekts gute Arbeit geleistet wurde.

11.6.4 Verbesserungs- und Erweiterungsmöglichkeiten

Die FriendsAround Komponente bietet die komplette Basisfunktionalität und erfüllt sämtliche Anforderungen eines grundlegenden FriendsAround Service, wie er für unser Projekt realisiert werden sollte. Dennoch besteht natürlich die Möglichkeit, diesen Service in Folgeprojekten zu verfeinern. Denkbar ist in diesem Zusammenhang beispielsweise eine Erweiterung des Datenschutzes beim Hinzufügen von Buddys in die FriendsAround Liste oder beim Auslesen von Profilattributen von Buddys. Außerdem könnte ein Informationsautomatismus entwickelt werden, der einem User meldet, sobald ein Buddy innerhalb eines definierten Radius auftaucht. Dies sind jedoch keine grundlegenden Eigenschaften des FriendsAround Service, diese wurden bereits im Laufe unseres Projekts vollständig entwickelt.

11.7 Communication-Komponente

11.7.1 Beschreibung

Die Communication-Komponente realisiert die Kommunikation verschiedener Clienten. Sie stellt als Core-Komponente die Methoden zur Verfügung, mit deren Hilfe die verschiedenen Applikationen den Benutzern eine nachrichtenbasierte Kommunikation ermöglichen kann. Zu diesem Zweck existieren zwei Nachrichtentypen, zum einen eine kurzlebige Nachricht, die direkt nach dem Empfang vom Server gelöscht wird, zum anderen eine persistente Nachricht, die erst nach explizitem Löschen verloren geht.

11.7.2 Realisierung

Die Communication-Komponente besteht aus einer SessionBean, die die Methoden zur Nachrichtenverwaltung zur Verfügung stellt, und einer EntityBean, die die Persistenz der langlebigen Nachrichten gewährleistet. Die SessionBean besitzt Methoden zum Senden und Empfangen von Nachrichten sowie dem Löschen persistenter Nachrichten. Als Parameter sind immer der Absender und der Applikationsname zu nennen, von dem der Methodenaufruf stattfindet. Kurzlebige Nachrichten werden mittels des BEA Weblogic JMS-Server realisiert. Hier wird pro Applikation ein so genanntes Topic erstellt und jedem Benutzer ein Subscriber zugeordnet. Dies ermöglicht eine schnelle Verarbeitung der Nachrichten, löscht diese allerdings, sobald alle zugeordneten Subscriber die Nachricht empfangen haben. Diese Möglichkeit der Nachrichtenübermittlung eignet sich gut für die Kommunikation zwischen unterschiedlichen Komponenten. Jedoch ist für einen Nutzer die Möglichkeit zur Abspeicherung von Nachrichten sehr wichtig.

Zu diesem Zweck haben wir nach dem zweiten Entwicklungsinkrement eine Möglichkeit zur Speicherung der Nachrichten vorgesehen. Diese wurde im weiteren Verlauf durch den Einsatz der EntityBean realisiert.
Da die ursprüngliche Planung noch keine Unterstützung von persistenten Nachrichten vorsah, konnte das Interface für die kurzlebigen Nachrichten nicht einfach um einen Parameter für die Persistenz erweitert werden, ohne bei mehreren anderen Komponenten Fehler zu verursachen. So wurden neue Methoden zum Senden und Empfangen von persistenten Nachrichten zum vorhandenen Interface hinzugefügt.

11.7.3 Probleme

Die Implementierung der kurzlebigen Nachrichten und damit der SessionBean verlief ohne große Probleme, einzig das Einrichten des JMS Servers barg einige kleinere Probleme. Probleme tauchten erst bei der Konfiguration des JMS-Servers auf. Es dauerte einige Zeit, bis eine funktionierende Einstellung der Parameter gefunden war. Hier half uns die Online-Dokumentation von Bea sehr weiter. Mehrfach geändert wurde auch der Filter, der es ermöglicht, dass ein Nutzer nur die für ihn bestimmten Nachrichten erhält.
Die Anforderungen nach einer persistenten Nachrichtenunterstützung kam erst später auf, so dass die Implementierung in zwei Schritten ablief. Ein erster Prototyp mittels direkter JDBC Verbindung zur Datenbank wurde schnell durch eine EntityBean Variante ersetzt, die nach Erstellen eines geeigneten Deploymentdeskriptors auch sehr gut funktionierte. Einzig die Möglichkeit, verschiedene Objekte an die Nachrichten anzuhängen, musste für die persistenten Nachrichten aufgegeben werden, da hier die Datenbank Probleme aufwarf, die in der Kürze der Zeit nicht zu lösen waren. Es lassen sich derzeit also nur Textnachrichten persistent machen.

11.7.4 Verbesserungs- und Erweiterungsmöglichkeiten

Erweiterungsmöglichkeiten für die Communication-Komponente bestehen. Möglich wäre eine flexiblere Unterstützung von Nachrichten, gerade der persistenten, denkbar. Hier wird zur Zeit nur das Senden, Empfangen und Löschen unterstützt. Außerdem wäre zum Beispiel die Einführung von Prioritäts- und Gelesen-Markern, sowie die Möglichkeit, nur ungelesene Nachrichten oder solche einer gewissen Priorität abzurufen denkbar. Ebenfalls möglich wäre eine Unterstützung von verschiedenen Nachrichtenordnern, für die spezielle Spam-Filter eingerichtet werden können oder deren Inhalt nach einer gewissen Zeit gelöscht wird. Dies setzt eine genauere Betrachtung des Absenders voraus sowie das Vorhandensein von Zeitstempeln von Nachrichten. Außerdem ist es sicher sinnvoll, die Nachrichten wieder um die Fähigkeit zu erweitern, beliebige Objekte anzuhängen. Gerade in Zeiten von Fotohandy und MMS sollte auf diese Option nicht verzichtet werden. Hierfür muss allerdings eine Lösung des Datenbankproblems gefunden werden.

11.8 Anwendung mCube.community

11.8.1 Beschreibung

Die Komponente mCube.community ist die Anwendungskomponente der ersten Schicht, auf der die weiteren Anwendungen, mCube.game, mCube.dating oder Anwendungen

von Drittanbietern basieren.
In unserem Schichtenmodell bestehen folgende Abhängigkeiten von mCube.community zu anderen Komponenten: nach oben zu den Anwendungen, nach unten zu Usermanagement, Routing, Groupmanagement, Locationing, Buddychat.
Neben der Funktion als Plattform für andere Anwendungen und der Bereitstellung der anwendungsübergreifenden Dienste des Logins und der Verwaltung der persönlichen Einstellungen und Daten bietet mCube.community eigenständige Funktionalitäten, wie Verwaltung einer Buddyliste, Chat und Nachrichtenaustausch unter Buddies.

11.8.2 Realisierung

Da es sich bei mCube um komponentenbasierte Entwicklung handelt, liegt auch der Schwerpunkt der Realisierung von mcube.community auf Enterprise Java Beans. Andererseits basiert die Realisierung auf dem Model-View-Controller-Konzept. Somit geschah bei der Implementierung die Einteilung in drei Bereiche: Enterprise Java Beans, CommandBeans und JSP.
Enterprise Java Beans CommunityManagementBean, CommunityLoginBean und CommunityBaseBean übernehmen als das Kernstück der Komponente die eigentliche Funktionalität der Community. Die JSP stellen die Fassade der Komponente dar. Durch diese wird die Benutzeroberfläche dargestellt und die vom User eingegebenen Daten und vorgenommenen Aktivitäten unter Benutzung des request-Objektes zur Verarbeitung ins Innere weitergeleitet. Die Weiterleitung übernimmt dabei die execute-Methode der CommandBeans, die an JSP anknüpfen.
Da die mCube-Anwendungen sowohl für das Handy als auch für das Web gedacht sind, sollen diese sowohl in HTML als auch in XHTML dargestellt werden können. Dazu wurde das MVC-Konzept auf XSLT umgestellt.
Logging wird mittels Log4J realisiert.

11.8.3 Probleme

Die meisten Probleme, die sich bei der Implementierung ergaben, waren nicht mCube.community spezifisch, sondern betrafen das gesamte Projekt. Dies gilt insbesondere für den Deployment-Prozess.
Es gab allerdings ein Problem, das nur Community betraf. Hierbei handelt es sich um die Implementierung des Buddychats. Anstatt die core-Komponente Chat nutzen zu können, mussten wir eine eigene, neue Komponente implementieren.
Implementierung mancher Anforderungen, z.B des Hinterlassens von Nachrichten an bestimmten Orten, wurde aufgrund von fehlender Reife der technischen Möglichkeiten oder deren Nicht-Einsatz in der Praxis (in diesem Fall genauer Ortungsverfahren) aufgegeben.

11.8.4 Verbesserungs- und Erweiterungsmöglichkeiten

Viele Funktionalitäten, die bei der Anforderungsanalyse mit Priorität „kann“ gekennzeichnet wurden, sind nicht, können aber implementiert werden. Dies gilt z.B für ein Shop-System mit Cubies.
Außerdem können bereits bestehende Dienste erweitert werden. Z.B kann neben des Buddychats ein Topicchat implementiert werden. Dieser kann sich zu einer Möglichkeit entwickeln, mit unbekannten Usern über ein bestimmtes Thema zu chatten und so neue Kontakte zu knüpfen und neue Freunde und Buddies zu gewinnen.

Desweiteren können detailliertere Profilangaben vom User abgefragt werden. Es kann dem User mehr Freiraum gestattet werden, indem ihm mehr Punkte geboten werden, die er sich selbst einstellen kann.

11.9 Anwendung mCube.game

11.9.1 Beschreibung

Die Anwendung mCube.game ist eine Anwendung, die auf mCube.community aufsetzt. Abhängigkeiten bestehen zu den core-Komponenten Usermanagement, Routing, Groupmanagement und Communication.
In der Anwendung mCube.game ist es möglich, ein Spiel zu spielen, zu den Spielregeln siehe Kapitel Anforderung und Kapitel Bedienungsanleitung.
Weitere Informationen zu dieser Anwendung finden sich auch im Kapitel Architektur.

11.9.2 Realisierung

Die Anwendung mCube.game wurde mit dem im Kapitel Architektur beschriebenen Model-View-Controller-Konzept und den ebenfalls dort beschriebenen Komponenten realisiert. Weitere Informationen finden sich also im Kapitel Architektur und im Anhang JavaDocs.
Logging wurde ebenfalls entsprechend implementiert.
Es konnte die Anforderung, Karten zu verkaufen und zu kaufen, nicht erfüllt werden, da in der Community kein entsprechendes Shop-System implementiert wurde. Alle weiteren Anforderungen konnten aber erfüllt werden.

11.9.3 Probleme

Die größte Herausforderung war zu Anfang, das Model-View-Controller - Konzept zu implementieren und es durch Entfernen aller Fehler funktionstüchtig zu machen. Weitere kleinere Probleme gab es mit dem Deployment der Enterprise Java Beans im Application Server, dies ist mit jedem neuen Application Server und jeder neuen Komponente eine neue Herausforderung. Als das geschafft war, gab es zunächst keine nennenswerten Schwierigkeiten mehr.
Mit Fortgang der Realisierung tauchten kleine Schwierigkeiten auf, die häufig mit den in unteren Schichten liegenden core-Komponenten zu tun hatten. So funktionierte der Messaging-Service der Communication-Komponente zu Beginn nicht hundertprozentig, da immer die erste gesandte Nachricht an einen neu erstellten Benutzer diesen nicht erreichte. Dieses Problem wurde aber ebenfalls schnell gelöst.
Der eingerichtete Build-Prozess erleichterte die Arbeit erheblich, da durch den automatisierten Deployment-Prozess die Anwendung mit einer viel größeren Geschwindigkeit im Application Server zu deployen war.
Eine Herausforderung wurde die Umstellung des Model-View-Controller - Konzeptes auf XSLT. Durch Änderungen an der Struktur des Model-View-Controller - Konzeptes funktionierte teilweise das Deployment nicht mehr. Als diese Strukturänderungen abgeschlossen waren, war es möglich, mit der Umstellung auf XSLT wie gewohnt fortfahren.
Bei der Erstellung des XSLT-Stylesheets für den Handy-Browser fiel auf, dass dieser zu viele Daten im Cache behielt. Beim Durchgang durch die Seiten stieß man auf alte, im

letzten Schritt eingegebene Daten. Hier half ein entsprechendes Tag, das den Browser veranlasst, nicht zu cachen.

11.9.4 Verbesserungs- und Erweiterungsmöglichkeiten

Eine Erweiterungsmöglichkeit ist eine Einführung eines Shop-Systems für Karten, damit diese ge- und verkauft werden können. Außerdem könnten weitere Möglichkeiten eingeführt werden, mit denen man an Karten gelangen kann.
Als etwas unpraktisch erwies sich während der Tests auch das System zum Abholen von Nachrichten. Hier könnte man in Zukunft eine Liste von Nachrichten einführen, unter denen man sich eine aussuchen kann, die man bearbeiten möchte.
Des Weiteren könnte die Anzahl der zur Auswahl stehenden Charaktere erweitert werden.
Eine gut zu vermarktende Erweiterung wäre die Möglichkeit, Karten in bestimmten Funkzellen unter bestimmten Umständen abholen zu können. So könnten Geschäfte, Kneipen und Kinos Anreize bieten, in ihr Haus zu kommen. Durch weitere Erweiterungen könnten so auch Stadttouren und „Schnitzeljagden“ implementiert werden.

11.10 Dating-Komponente

11.10.1 Beschreibung

Die Dating Komponente ist eine Komponente der Anwendungsschicht, die sich auf die Komponenten der core-Schicht stützt.Die Funktionalität der Dating-Komponente besteht darin, Datingpartner zu suchen, zu lokalisieren und anschließend zusammenzubringen. Weiterhin bietet die Komponente Funktionalitäten zur Profilerstellung, zum Nachrichtenaustausch zwischen den Datingbenutzern und zur Verwaltung der Buddylisten.

11.10.2 Realisierung

Die Realisierung der Dating-Komponente wurde in drei Bereiche aufgeteilt; Enterprise Java Beans, CommandBeans und Java Server Pages.

11.10.2.1 Enterprise Java Beans

Die zentralen Aufgaben der Dating Komponente übernimmt das DatingBean, das als stateful SessionBean implementiert wurde, so dass es eine Instanz dieser Komponente pro User gibt. Dazu wird bei der Erzeugung des Beans der Methode ejbCreate eine UserID übergeben, die lokal gespeichert wird. Alle weiteren Methodenaufrufe werden daraufhin ausgehend von diesem User betrachtet. Da für die Realisierung des Dating Service die Funktionalität der core Komponenten Communication, Usermanagement, Matching und Groupmanagement benötigt wird, wird in der Methode ejbCreate eine Verbindung zu diesen Komponenten etabliert.
Um die oben genannten Funktionalitäten der Listenverwaltung zu realisieren, wurden vier Methoden implementiert, die zur Erstellung, zum Entfernen, zur Blockierung und zum Freigeben von Buddylisten bzw. Blockedlisten dienen. Dazu wurden die Funktionalitäten von der Usermanagement bzw. Groupmanagement Komponente erweitert und entsprechend genutzt. Weiterhin wurden Methoden der Profilverwaltung implementiert. Zur Erfüllung der Lokalisierungsanforderung wurden Methoden implementiert,

die die core-Komponenten nutzen. Hier wurden besonders die Methoden der Locationing bzw. Routing Komponenten verwendet.
Es wurden auch Methoden für Nachrichtenaustausch entwickelt, die Messages zwischen zwei Datingusern ermöglichen; dabei wurden hauptsächlich die Methoden der Communication-Komponente genutzt und weiter ans Dating angepasst. Logging wurde integriert.

11.10.2.2 CommandBeans

Die CommandBeans dienen als Datenaustausch-Brücke zwischen dem DatingBean und den Java Server Pages. Alle CommandBeans haben eine execute() Methode, die dem Datenaustausch dient. Hauptsächlich wird in der execute Methode das Request-Objekt mit Attributen und Parametern gesetzt, die dann in den JSP-Seiten ausgelesen und ausgewertet werden können, außerdem werden die Methoden des DatingBean aufgerufen. Weiterhin können die CommandBeans auf neue CommandBeans verweisen, was die Handhabung der Beans vereinfacht.

11.10.2.3 Java Server Pages

Alle JSP-Seiten der Dating-Komponente enthalten zum großen Teil XML-Code, der zur Vereinfachung der Formattransformation dient. Es wurden für das HTML- und xHTML-Format die entsprechenden Transformationsregeln in die XSL-Dateien geschrieben. Die Hauptfunktionalität der JSP-Seiten des Dating besteht darin, die Parameter bzw. Attribute aus dem Request-Objekt auszulesen und mit Hilfe von XML-Tags zu beschreiben bzw. auszuwerten. Diese XML-Tags werden dann mit Hilfe von XSLT-Prozessoren in das HTML- oder xHTML-Format umgewandelt.

11.10.3 Probleme

Bei der Implementierung der Dating Komponente traten keine größeren Probleme auf. Es wurden jedoch Anpassungen im Laufe des Entwicklungsprozesses vorgenommen bzw. die Architektur wurde leicht verändert (auf die DatingEntityBean wurde ganz verzichtet), was zu leichter Veränderung der Implementierung führte.

Kapitel 12

Installation

Dieses Kapitel beschreibt die zum Betrieb der mCube - Plattform notwendigen Software Komponenten sowie deren Installation / Konfiguration. Ausgegangen wird von der Installation auf einer Windows Plattform.

12.1 Verwendete Software

Es folgt eine kurze Vorstellung der eingesetzten Komponenten. Eine genauere Beschreibung der notwendigen Installationsschritte findet in den folgenden Abschnitten statt.
Folgendende Komponenten wurden verwendet:

BEA Weblogic Application Server Eingesetzt wurde Version 7.0. Die Kompatibilität zu vorherigen Versionen bzw. zu Application Servern anderer Hersteller wurde nicht getestet. Benötigt wird ein Server, der mindestens folgende Technologien unterstützt:

- J2EE 1.3
- CMP 2.0

Oracle Database 8*i* Eingesetzt zur Persistierung von Entity-Bean- / sowie der JMS - Daten.

Netscape Directory Server 6.0.1 Eingesetzt zur Persistierung der User-Profile.

MCUBE.EAR Enthält die Business Logik der MCUBE-Plattform in Form eines `EnterpriseARchives`, wie im `J2EE` Standard spezifiziert. Die proprietären Deskriptoren wurden für die Verwendung im `BEA Weblogic` Server erstellt. Die Verwendung in einem anderen Kontext ist nur nach gesonderter Anpassung möglich.

Diverse Java Libraries Utility Komponenten, die zur korrekten Ausführung der Plattform auf dem Application Server benötigt werden.

12.2 Installation des BEA Weblogic Servers

Alle notwendigen Programme finden sich auf der mCube Installations-CD im Ordner \BEA Weblogic 7.0
Notwendige Schritte:

1. Ausführen der Datei `weblogic700_win.exe`.
2. Standardeinstellungen der Installation beibehalten.
3. Ausführen des `BEA Configuration Wizard`.
4. Anlegen einer neuen Domain. (`MCUBE`).

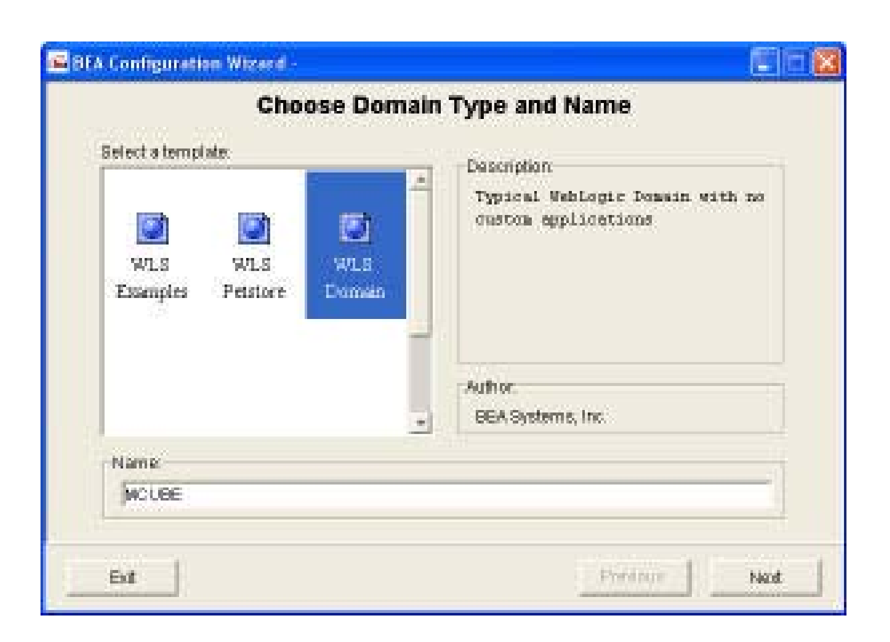


Abbildung 12.1: Anlegen einer neuen Domain

Obwohl noch einige weitere Einstellungen am Application Server zu tätigen sind, wird jetzt zuerst die Installation der Oracle Datenbank beschrieben, da sich die dort vorgenommenen Einstellungen teilweise mit denen des Servers bedingen.

12.3 Installation der Oracle Datenbank

Alle notwendigen Programme finden sich auf der mCube Installations-CD im Ordner \Oracle 8i.
Notwendige Schritte:

1. Entpacken der Datei `nt817personal.zip` in ein temporäres Verzeichnis.
2. Ausführen der Datei `setup.exe`.

3. Standardeinstellungen der Installation beibehalten.
4. Das Datenbankskript `dbschema.sql` ausführen.

12.4 Konfiguration des BEA Weblogic Servers

Zur Konfiguration des Servers muss die Administrationskonsole aufgerufen werden. Diese findet sich, wenn man die Standardeinstellungen bei der Installation beibehalten hat, als HTML-Oberfläche unter `http://IP-Adresse:7001/console`. Folgende Konfigurationen sind vorzunehmen:

Einrichten eines JDBC-Connection Pools

- Name: `OraclePool`
- URL: `jdbc:oracle:thin@IP-Adresse:1521:Oracle`
- Driver: `oracle.jdbc.driver.OracleDriver`

Einrichten einer Transaction Datasource (TX)

- Name: TXOracleSource
- Poolname: OraclePool

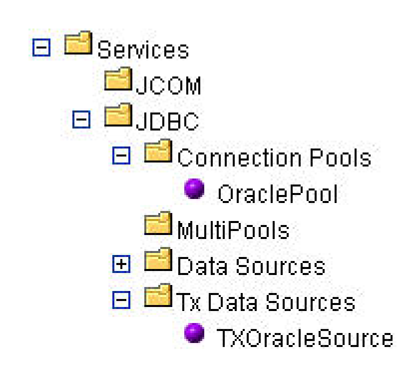


Abbildung 12.2: JDBC Einstellungen

JMS Konfiguration

- Einrichten eines `JMSServers` mit Namen `MyJMSServer`.
- Einrichten einer `ConnectionFactory` mit Standardeinstellungen.
- Einrichten der Destination `Mcube_Community`.
- Einrichten der Destination `Mcube_Dating`.
- Einrichten der Destination `Mcube_Game`.

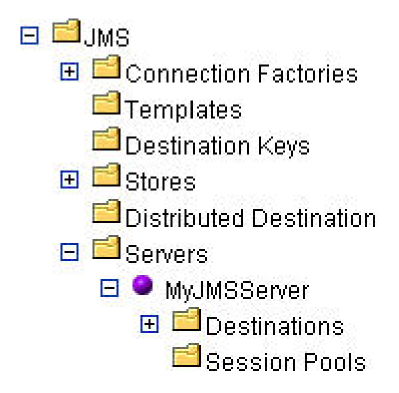


Abbildung 12.3: JMS Einstellungen

12.5 Installation des Netscape Directory Servers

Alle notwendigen Programme / Dateien finden sich auf der mCube Installations-CD im Ordner `\Netscape Directory Server 6.01`.
Notwendige Schritte:

1. Entpacken der Datei `d601diu.zip`.
2. Ausführen der Datei `setup.exe`.
3. Standardeinstellungen der Installation beibehalten.

12.5.1 Erläuterungen zum LDAP-Server

Zur erfolgreichen Inbetriebnahme des Servers mit der mCube Plattform ist die Einrichtung einer speziellen Datenbankstruktur erforderlich. Da diese nur mühsam per Hand vorzunehmen ist, wird auf der CD die Datei `LDAPImport.LDIF` mitgeliefert, die in der Administrationskonsole des Servers importiert werden kann. Die Konfiguration ist damit beendet.



Abbildung 12.4: Netscape Directory Server DB Import

Zum besseren Verständnis folgt trotzdem eine kurze Beschreibung der verwendeten Struktur. Die interne Struktur des Servers ist grob in zwei Bereiche aufgeteilt:

USER Enthält die gespeicherte Daten eines User in Attributform. (z.B. Alter, Geschlecht...)

MCUBECONTAINER Enthält die Metadaten über die Profilzugehörigkeit eines speziellen Attributes. Zur Zeitpunkt der Dokumentationserstellung existieren folgende Profile:

- `.game`
- `.community`
- `.dating`

Da in `USER` alle Attribute gespeichert werden, kann durch die Zuordnung der Attribute zu dedizierten Profilen in `MCUBECONTAINER` eine selektive Anzeige angestoßen werden. Die Daten in `MCUBECONTAINER` fungieren damit quasi als Zugriffsmaske.

12.5.1.1 Attribut - Profil Zuordnungen

Profil: **.game**

cn	Game
ou	MCubeContainer
dc	mcu
dc	be

Attribute

- battlehistory
- constitution
- dextertity

- eventcards
- intelligence
- itemcards
- level
- lifepoints
- maxdistance
- numberofimproves
- strength
- usualcards
- weaponcards
- charactertype

Profil: **.dating**

cn	Dating
ou	MCubeContainer
dc	mcu
dc	be

Attribute

- haircolor
- height
- searchradius
- smoker
- weight

Profil: **.community**

cn	Community
ou	MCubeContainer
dc	mcu
dc	be

Attribute

- sn
- cn
- givenName
- mobile

- profiles
- birthdate
- city
- cubies
- description
- email
- friendsaroundlist
- isLoggedIn
- isValidated
- plz
- positioningallowed
- sex
- street
- telephonenumber

12.6 Externe Java Libraries

Zur korrekten Funktionsweise benötigt die mCube Plattform diverse Utility Komponenten, die auf der Installations-CD im Ordner `\libs` mitgeliefert werden.
Die Installation / Konfiguration dieser Komponenten beschränkt sich auf das Kopieren aller Dateien aus o.g. Verzeichnis in den `\lib` Ordner des in den vorherigen Schritten installierten Application Servers.



Beispielpfad: `\bea\weblogic700\server\lib`

Kapitel 13

Benutzerhandbuch

Das Benutzerhandbuch gibt einen detaillierten Überblick über die Funktionen der drei Bereiche mCube.community, mCube.game und mCube.dating, verbunden mit einem exemplarischen Durchlauf eines Spiels bzw. einer Partnersuche.

13.1 mCube.community

Die Community dient als Eingangsportal zu den von mCube angebotenen Anwendungen. Dies bedeutet, dass die in der Community vorgenommene Anmeldung ebenso wie alle anderen Einstellungen in allen Anwendungen gültig sind. Die mCube-Anwendungen sind aus der Community über die Links auf dem Bildschirm erreichbar, allerdings erst nach der Anmeldung.
Daneben bietet Community eigenständige Dienste wie Verwaltung einer Buddyliste und Dienste um Buddys rund herum: Auskunft über FriendsAround, d. h. Buddys, die sich zum aktuellen Zeitpunkt in der Nähe des betroffenen Users befinden, Chat mit Buddys, Möglichkeit, Nachrichten an Buddys zu versenden und Nachrichten von Buddys zu empfangen. Ebenso werden von der mCube-Community Dienste zur Administration der Userdaten zur Verfügung gestellt.

13.1.1 Neuanmeldung

Der erste Schritt in der Community ist die Neuanmeldung des Users. Dabei werden vom User folgende Angaben benötigt. Er soll seinen richtigen Vor- und Nachnamen eingeben, die Handynummer, mit der er mCube-Dienste in Anspruch nehmen wird, sein Geburtsdatum und die Postleitzahl seines Wohnortes. Neben realen Daten sollen solche eingegeben werden, die nur für mCube von Bedeutung sind. Hierbei handelt es sich um einen Nickname und ein Passwort, das zur Sicherheit wiederholt eingegeben werden soll. Nach der Eingabe der Daten geschieht die erste Anmeldung in das System automatisch.

13.1.2 Login

Bei der Anmeldung in das System werden vom User nur sein Nickname und sein Passwort erwartet. Soll mindestens eins von beiden falsch eingegeben werden, so erscheint der gleiche Bildschirm noch einmal, allerdings mit einer Fehlermeldung. Damit

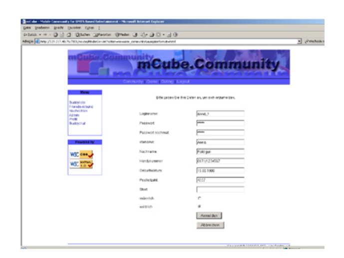


Abbildung 13.1: Neuen Benutzer anmelden

wird der User aufgefordert, die für die Anmeldung erforderlichen Daten noch einmal einzugeben.



Abbildung 13.2: Einloggen

13.1.3 Buddyliste

Der erste Bildschirm, der dem User nach dem Einloggen erscheint, ist die Buddyliste des Users. Über diesen wird die Verwaltung der Buddyliste erledigt.



Abbildung 13.3: Buddyliste

Es werden die Namen aller Buddys aufgelistet, ebenso die Information, ob sich der betreffende Buddy gerade online oder offline befindet. Der Status wird insbesondere durch Farben hervorgehoben. Durch grün wird markiert, dass sich der entsprechende Buddy online befindet, rot bedeutet, dass der Buddy offline ist.
Des Weiteren befindet sich neben einem jeden Buddynamen ein Schlüssel-

oder Schlosssymbol . Dieses deutet an, ob der Buddy dem User Nachrichten senden darf.

Durch einen Klick auf den Buddynamen werden detaillierte Informationen über den Buddy angezeigt, und zwar seine Vor- und Nachnamen. Allerdings sind Buddydetails nur dann sichtbar, wenn der jeweilige Buddy die Erlaubnis dazu gegeben hat, seine persönlichen Daten anderen Usern zugänglich zu machen.
Natürlich ist es möglich, die Buddyliste zu verändern, d.h. bestehende Buddys aus der Liste zu entfernen und neue Buddys in die Liste hinzuzufügen. Das Löschen der Buddys funktioniert auf folgende Art und Weise, nämlich durch einen Klick auf das Löschsymbol , das sich neben dem Namen eines jeden Buddy befindet.

Hinzugefügt wird ein Buddy, indem auf den Link „Buddy hinzufügen“ geklickt und anschließend der Name des neuen Buddys eingegeben wird.



Abbildung 13.4: neuen Buddy hinzufügen

Weitere Funktionalitäten des Systems sind über die Menüleiste, links vom Hauptbildschirm, erreichbar.

13.1.4 FriendsAround

Auf diesem Bildschirm werden alle Buddys des aktuellen Users aufgelistet, die zum aktuellen Zeitpunkt online sind. Neben dem Namen jedes Buddy steht dessen Entfernung vom User sowie die Himmelsrichtung, in der er sich nun befindet.



Abbildung 13.5: Friends Around

13.1.5 Nachrichten

Die mCube-Community bietet eine Möglichkeit, Nachrichten mit Buddys auszutauschen. Dies bedeutet, ein User kann jedem seiner Buddys eine Nachricht schreiben und von seinen Buddys Nachrichten empfangen, falls diese von ihm dazu berechtigt worden sind (Schlüssel beim Buddy auf dem Bildschirm „Buddyliste" eingestellt).

Über den Menüpunkt „Nachrichten" gelangt man an ein Bildschirm, von dem aus der User neue und gespeicherte Nachrichten lesen und Nachrichten schreiben kann.

Das Lesen von Nachrichten geschieht wie gewohnt durch das Anklicken der betreffenden Nachrichten.



Abbildung 13.6: Nachrichten auswählen

Beim Schreiben von Nachrichten wird aus der Liste aller Buddys der Empfänger ausgewählt, der Text der Nachricht in das dafür vorgesehene Feld geschrieben und die Nachricht anschließend versendet.

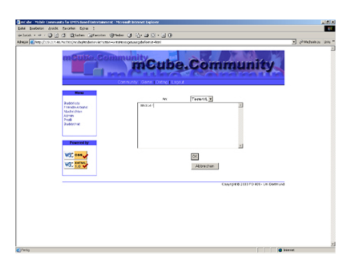


Abbildung 13.7: Nachricht schreiben

13.1.6 Admin

Über den Menüpunkt „Admin" kann man die mCube betreffenden Einstellungen ändern. Hierbei handelt es sich um die Erlaubnis, Detaildaten weiter zu geben. Dies bedeutet, dass der User mit dieser Einstellung angeben kann, ob er in der Buddyliste anderer User mit detaillierten Daten auftreten möchte oder ob er die Anzeige seines Vor- und Nachnamens verbietet und nur sein Nickname freigibt.

Des Weiteren kann unter diesem Menüpunkt das Passwort geändert werden. Dafür muss das jetzige Passwort eingegeben werden und anschließend zweimal das neue Passwort.



Abbildung 13.8: Administration der persönlichen Einstellungen

13.1.7 Profile

Die persönlichen Daten des Users können unter dem Menüpunkt „Profile“ geändert werden, und zwar der Vor- und der Nachname des Users.



Abbildung 13.9: persönlicher Profil

13.1.8 Buddychat

Der Buddychat ermöglicht eine weitere Art der Kommunikation unter den Buddys. Alle Buddys können am Chat teilnehmen. Allerdings sollte darauf geachtet werden, dass Chatten nur dann reibungslos funktioniert, wenn von zwei Usern jeder jeweils in der Buddyliste des anderen eingetragen ist.
Die Teilnahme am Buddychat funktioniert wie gewohnt. Der User schreibt eine Nachricht und sendet diese. Anschließend kann man die gesamte Kommunikation in dem dafür vorgesehenen Feld betrachten, wenn man auf „Nachrichten abholen“ klickt.



Abbildung 13.10: Buddychat

13.2 mCube.game

Die Anwendung mCube.game verbindet die Konzepte von Sammelkartenspielen und rundenbasierten Kampfspielen. Sie basiert auf erweiterbaren Charakteren, die gegeneinander kämpfen können. Die Darstellung ist sowohl auf einem normalen Webbrowser als auch auf einem Handy möglich.

13.2.1 Hauptbildschirm und Navigation



Abbildung 13.11: Hauptbildschirm von mCube.game in einem Webbrowser



Abbildung 13.12: Hauptbildschirm von mCube.game in einem Handybrowser

Man gelangt über den Menüpunkt game der mCube.community in das Spiel. Nach der Erklärung des Hauptbildschirms und der Navigation folgt ein beispielhafter Durchlauf durch das Spiel. Genauere Ausführungen zu den Regeln finden sich in der Anforderungsanalyse.
Da mCube.game auf mehreren Endgeräten läuft, gibt es auch mehrere Oberflächen. Hier werden die Handyoberfläche und die Browseroberfläche betrachtet. Der Hauptunterschied in der Navigation ist lediglich, dass man auf dem Handy weniger Links zur

Navigation zur Verfügung hat. Auf dem Handy (siehe Abbildung 13.12) befinden sich auf jeder Seite Links zu der Startseite der Community, zum Logout und zur Startseite von Game. Alle anderen Links dienen der Logik des Spiels. Des Weiteren ist es auf dem Handy immer möglich, eine Seite zurück zu gehen. Auf einem normalen Webbrowser (siehe Abbildung 13.11) hat man zusätzlich noch die Möglichkeit, gezielt zu vorhergehenden Seiten zu gehen (Verlauf im unteren Teil des Menüs).
Die Menüpunkte im Hauptbildschirm sind „Nachricht abholen“, „neuer Charakter“, „Charakter bearbeiten“ und „Historie“. Diese Punkte werden im Folgenden in einem exemplarischen Durchlauf erläutert. Des Weiteren sieht man noch eine Liste mit Spielern, die online sind. Diesen kann man Nachrichten schicken.
Nun beginnt der exemplarische Durchlauf. Solange klar ist, welchen Link man verfolgen bzw. welchen Button man drücken muss, ist dies nicht explizit erwähnt. Der exemplarische Durchlauf orientiert sich an der Version für den Webbrowser. Die Elemente im Handybrowser sind entsprechend vorhanden.

13.2.2 Erstellen eines neuen Charakters

Bevor Sie spielen können, müssen Sie einen Charakter erstellen, der Sie während des Spiels begleitet. Dazu gibt es im Menü den Punkt „neuer Charakter“. Dort stehen Ihnen vier Charaktere zur Auswahl.



Abbildung 13.13: Auswahl eines Charaktertyps für den neuen Charakter

Jeder dieser Charaktere hat eigene Stärken und Schwächen, die sich in den Attributen und den Startkarten ausdrücken.

Charakter	Konstitution	Geschicklichkeit	Stärke	Intelligenz
Elf	3	7	4	6
Magier	3	6	3	8
Barbar	7	3	7	3
Paladin	4	5	6	5

Tabelle 13.1: Werte der Charaktertypen

Die Startkarten der Charaktere sind:

Elf

Ausrutschen	Wirkung:	Gegner fällt hin, 3 Schadenspunkte Gegner
Amulett der Geschicklichkeit	Besonderheit:	Geschicklichkeit+1

Ritzkihutu	abhängig von:	2×Geschicklichkeit
	Offensivbonus:	3
	Defensivbonus:	3
	Minwert:	2
	Maxwert:	5
	Einschränkung:	Nur Elf

Magier

Heiltrank	Wirkung:	8 Schadenspunkte zurück
Feuerball	abhängig von:	2×Intelligenz
	Offensivbonus:	3
	Defensivbonus:	0
	Minwert:	1
	Maxwert:	10
	Einschränkung:	Magier
Amulett der Intelligenz	Besonderheit:	Intelligenz+1
Stab	abhängig von:	Stärke, Geschicklichkeit
	Offensivbonus:	0
	Defensivbonus:	5
	Minwert:	1
	Maxwert:	3
	Einschränkung:	Nur Magier

Barbar

Ausrutschen	Wirkung:	Gegner fällt hin, 3 Schadenspunkte Gegner
Lederrüstung	Defensivbonus:	3
	Besonderheit:	Geschicklichkeit-1
Keule	abhängig von:	2×Stärke
	Offensivbonus:	5
	Defensivbonus:	0
	Minwert:	1
	Maxwert:	10
	Einschränkung:	Stärke $\geq$ 7

Paladin

Schwert	abhängig von:	Stärke, Geschicklichkeit
	Offensivbonus:	2
	Defensivbonus:	2
	Minwert:	4
	Maxwert:	6
	Einschränkung:	Stärke $\geq$ 6
Eisenrüstung	Defensivbonus:	5
	Besonderheit:	Geschicklichkeit-2
Mauer	Defensivbonus:	10
	Einschränkung:	Elf, Magier, Paladin

Angenommen Sie entscheiden sich hier für einen Barbaren, so können Sie ihm einen passenden Namen wie z.B. „Olli Kahn“ geben.



Abbildung 13.14: Eingabe des Namens des neuen Charakters

Nachdem Sie den Namen eingegeben haben, bekommen Sie eine Statusmeldung, dass Ihr neuer Charakter erstellt wurde.

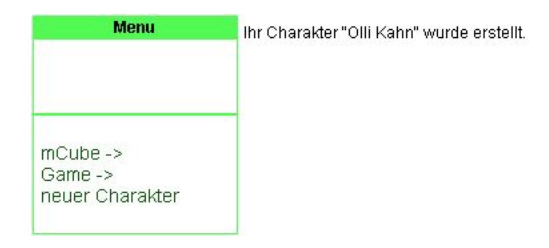


Abbildung 13.15: Der neue Charakter wurde erstellt

13.2.3 Charakter bearbeiten

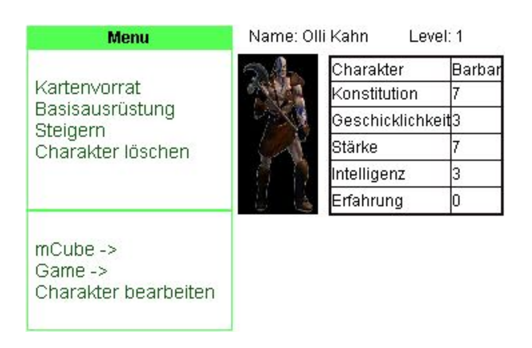


Abbildung 13.16: Menüpunkt „Charakter bearbeiten“

Unter dem Punkt „Charakter bearbeiten“ des Hauptmenüs können Sie sich nun ihren neuen Charakter angucken. Sie haben hier die Möglichkeit, sich den „Kartenvorrat“ anzugucken, die „Basisausrüstung“ festzulegen, Steigerungen vorzunehmen, falls Sie einen höheren Level erreichen, und ihren Charakter wieder zu löschen.

13.2.3.1 Kartenvorrat

Im Menüpunkt „Kartenvorrat“ sehen Sie alle Karten des Charakters. Diese sind eingeteilt in Gegenstände, Waffen und Ereignisse. Diese Einteilung sagt Ihnen, wann Sie die Karten benutzen können. Gegenstände können Sie als Basisausrüstung an sich tragen, falls Ihr Charakter dazu in der Lage ist. Waffen können Sie im Kampf wählen und dort benutzen. Ereignisse können Sie einmal pro Kampf anwenden.



Abbildung 13.17: Kartenvorrat des Charakters

Sollten Sie sich für Ihre neue Waffe interessieren, können Sie sich angucken, was eine Keule so alles kann.

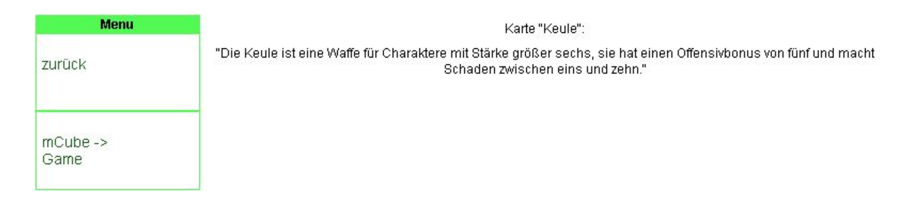


Abbildung 13.18: Ansicht der Karte „Keule“

13.2.3.2 Basisausrüstung

Die Basisausrüstung trägt ein Charakter immer bei sich. Die Modifikationen zeigen also immer Wirkung.



Abbildung 13.19: Basisausrüstung des Charakters

Wollen Sie z.B. Ihre „Lederrüstung“ immer tragen, damit Ihre Defensive gestärkt ist, so wählen Sie die Karte aus und fügen sie mittels „add“ Ihrer Basisausrüstung hinzu.

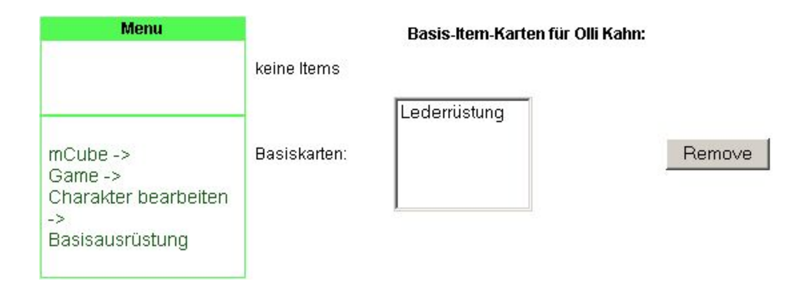


Abbildung 13.20: Erweiterte Basisausrüstung des Charakters

13.2.3.3 Steigern von Attributen

Da Sie noch keine Attribute steigern können, können Sie an diesem Punkt noch nichts tun.

13.2.3.4 Charakter löschen

Aus Interesse schauen Sie sich noch den Punkt „Charakter löschen“ an, der zur Sicherheit einen weiteren Link „Charakter endgültig löschen“ enthält.



Abbildung 13.21: Löschen des Charakters

13.2.4 Nachrichten an andere Spieler

Nachdem Sie Ihren Charakter nun etwas näher kennen, können Sie Kontakt zu anderen Spielern aufnehmen. Dazu muss man einfach auf einen Spieler klicken, der online ist. In diesem Beispiel ist dies der Spieler „Zwölf“.
Sie haben drei Nachrichtentypen, die Sie verschicken können. Eine Standardnachricht, ein Tauschangebot und eine Kampfaufforderung.

13.2.4.1 Standardnachricht

Da Sie den fremden Spieler nicht überrumpeln wollen, fragen Sie vielleicht erstmal höflich an, ob er zum Tauschen und Kämpfen bereit wäre.
Das System teilt Ihnen daraufhin mit, ob das Versenden der Nachricht erfolgreich war. Ob eine Nachricht für Sie vorliegt, können Sie im Hauptmenü mit dem Punkt „Nachricht abholen“ überprüfen. Sollte eine Nachricht vorliegen, so wird sie sofort angezeigt. Glücklicherweise antwortet Spieler „Zwölf“ schnell und ist zum Tauschen und Kämpfen bereit.

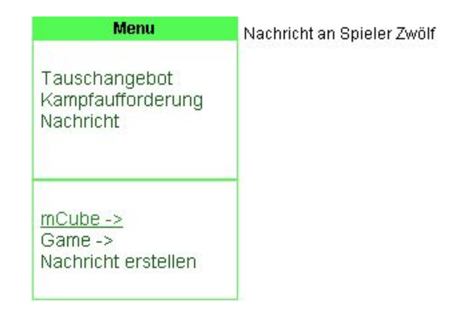


Abbildung 13.22: Nachricht an Spieler Zwölf



Abbildung 13.23: Absenden einer Standardnachricht

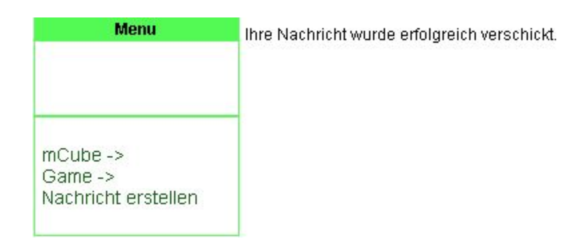


Abbildung 13.24: Status der Nachricht

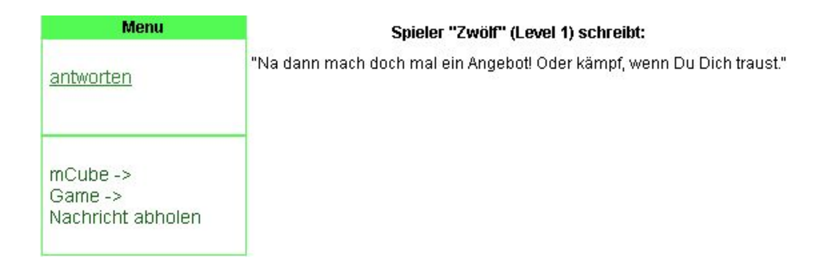


Abbildung 13.25: Antwort von Spieler Zwölf

13.2.4.2 Tauschangebot

Als erstes wollen Sie möglicherweise sehen, ob Sie von ihm eine gute Karte erhalten können. Wenn Sie auf „Tauschangebot“ gehen, dann sehen Sie zunächst, welche Karten Sie anbieten können und welche Sie vom Spieler erfragen können.
Beispielsweise bieten Sie dem Spieler „Ausrutschen“ an (Button „+“), da sich das für

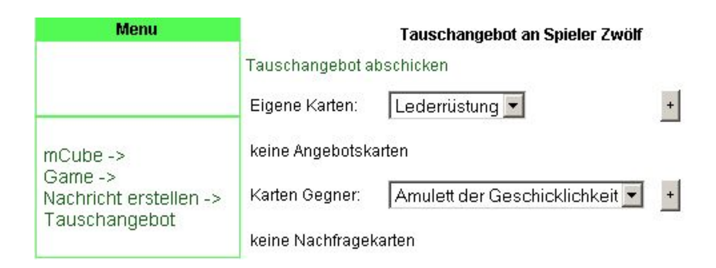


Abbildung 13.26: Tauschangebot an Spieler Zwölf

einen Barbaren nun wirklich nicht gehört und wollen dafür das „Amulett der Geschicklichkeit“ (Button „+“), das Ihren Charakter ein wenig geschickter machen würde.



Abbildung 13.27: Das fertige Tauschangebot

Für zukünftige Tauschaktionen können Sie auch noch einen kurzen Blick darauf werfen, wie das Tauschangebot aussieht, wenn es bei Spieler „Zwölf“ ankommt (Abbildung 13.28).

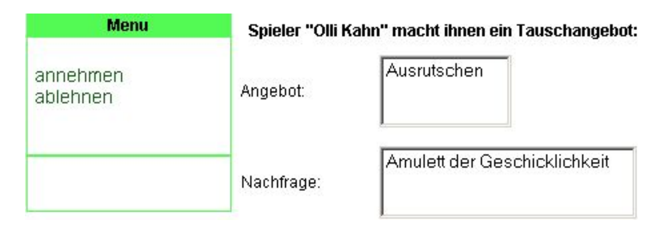


Abbildung 13.28: So empfängt Spieler Zwölf das Tauschangebot

Glücklicherweise nimmt er es an, und so können Sie bald die Nachricht abholen, dass der Tausch durchgeführt wurde.

13.2.4.3 Kampfaufforderung

Nach dieser friedvollen Transaktion wird es jetzt allerdings Zeit für einen Kampf. Zu diesem Zwecke formulieren Sie eine Kampfaufforderung.

13.2.5 Kampf zweier Charaktere

Nimmt der andere Spieler die Kampfaufforderung an, so gelangen Sie beim Abholen von Nachrichten auf die Auswahl unserer Basisausrüstung.

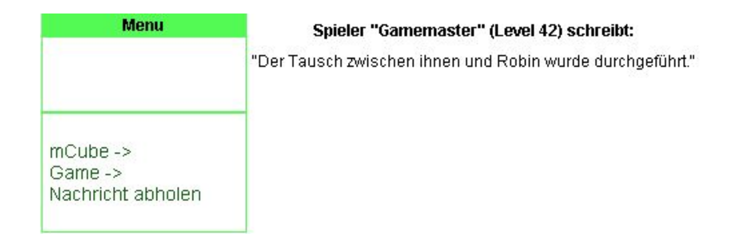


Abbildung 13.29: Der Tausch wurde durchgeführt

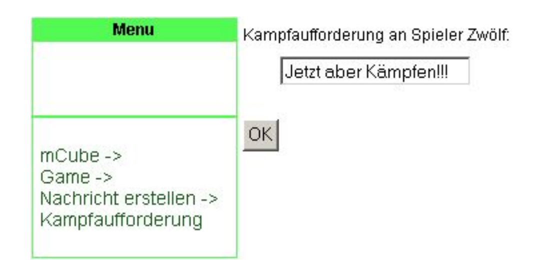


Abbildung 13.30: Kampfaufforderung an Spieler Zwölf

13.2.5.1 Wahl der Basisausrüstung

Nun können Sie ein letztes Mal vor dem Kampf die Basisausrüstung bestimmen.

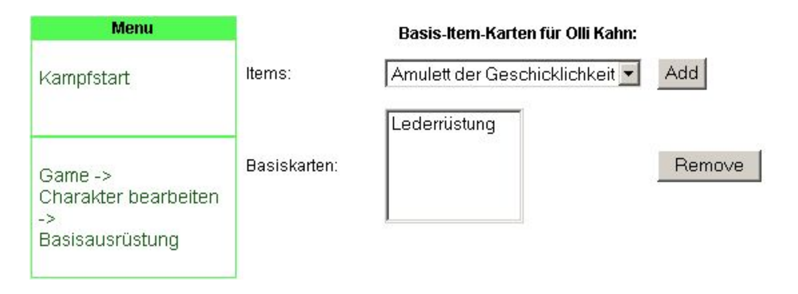


Abbildung 13.31: Basisausrüstungswahl vor dem Kampf

Natürlich probieren Sie gleich Ihre neue Karte aus und fügen sie Ihrer Basisausrüstung hinzu. Danach geht es aber zum „Kampfstart“.

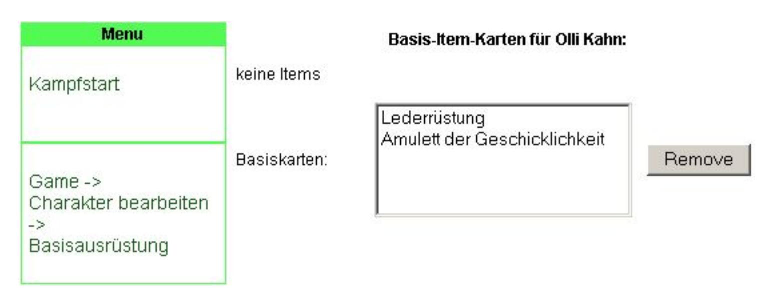


Abbildung 13.32: Die neue Karte bei der Basisausrüstung

13.2.5.2 Eine Kampfrunde

Menu
Kampf ihres Charakters Olli Kahn gegen Zwölf
Lebensenergie: 35
Der Kampf beginnt! Wähle Deine Taktik.
Weapons: Keule
Sie haben keine Events
Offensivität: 98
Kampfspruch: Auffe Omme!!!
OK

Abbildung 13.33: Wahl der Taktik für die erste Kampfrunde

In der Kampfrunde können Sie nun Ihre Taktik festlegen. Sie müssen angeben, wie offensiv Sie vorgehen wollen. Je offensiver man angreift, desto mehr Schaden richtet man zwar an, desto mehr Schaden kann man aber auch erleiden, da man dann seine Defensive offen lässt. Als Barbar stört Sie das natürlich nicht und Sie gehen auf eine Offensive von 98. Als Waffe nehmen Sie Ihre Keule und Ereignisse haben Sie leider nicht mehr. Nun gilt es noch einen markigen Kampfspruch einzutragen. Sie entscheiden sich für „Auffe Omme!!!“.

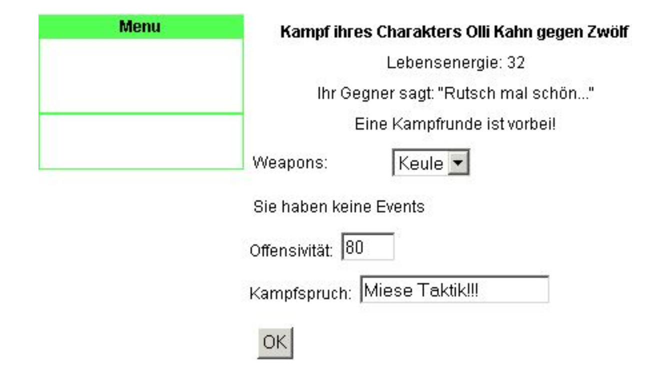


Abbildung 13.34: Zweite Kampfrunde und Ergebnis der vorherigen Runde

Hat der andere Spieler ebenfalls seinen Zug vollendet, gelangt man über „Nachricht abholen“ in die zweite Kampfrunde.

Sie sehen hier, dass Sie drei Lebenspunkte verloren haben, da Ihr feiger Gegner offensichtlich „Ausrutschen“ eingesetzt hat. Diesmal gehen Sie etwas vorsichtiger zu Werke und beschweren sich noch über die miese Taktik. Dies geht weiter, bis einer der Charaktere keine Lebensenergie mehr hat.

13.2.5.3 Kampfergebnis

Nachdem einer der Charaktere keine Lebensenergie mehr hat, ist der Kampf beendet und der Sieger wird bekannt gegeben. Leider haben Sie Ihren ersten Kampf verloren.

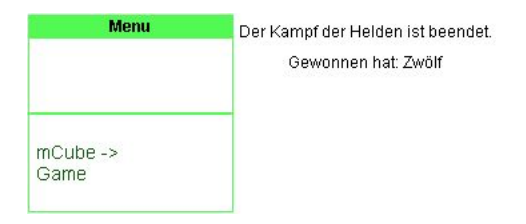


Abbildung 13.35: Das Kampfergebnis

13.2.6 Historie eines Charakters

In der Historie können Sie sich noch einmal Ihre Kämpfe ansehen. Da der Gegner rot markiert ist, werden Sie nochmal an Ihre Schmach erinnert. Immerhin können Sie sich jedoch über fünf Erfahrungspunkte freuen.



Abbildung 13.36: Die Historie des Charakters

13.2.7 Levelaufstieg

Durch den Kampf von „Olli Kahn“ und „Zwölf“ hat Zwölf 50 Erfahrungspunkte überschritten und steigt so zum ersten Mal einen Level auf.



Abbildung 13.37: Menüpunkt „Charakter bearbeiten“ des Charakters Zwölf

Beim Steigern sieht man nun, dass „Zwölf“ durch den Levelgewinn einmal steigern darf. Als Elf steigert Zwölf natürlich seine Geschicklichkeit.



Abbildung 13.38: Steigern eines Charakterwerts



Abbildung 13.39: Resultat der Steigerung

13.3 mCube.dating

Die Anwendung mcube.dating ermöglicht das Auffinden von Partnern aufgrund von spezifizierbaren Profilen. Es ist möglich, sich einander Nachrichten zu schicken und sich zu lokalisieren. Die Darstellung ist sowohl auf einem normalen Webbrowser als auch auf einem Handy möglich.
Man gelangt über den Menüpunkt Dating der mCube.community in das Hauptmenü. Standardmäßig wird hier eine Liste bereits ausgewählter Partner angezeigt. Im Folgenden werden die einzelnen Sektionen in eigenen Abschnitten behandelt. Am Ende wird mit einem Beispiel in die Benutzung der Applikation eingeführt. (siehe 13.3.6 „Wie man einen Partner findet“)

13.3.1 Hauptbildschirm und Navigation



Abbildung 13.40: Das Hauptmenü

13.3.2 Das eigene Profil

Nachdem zum ersten Mal die Dating Anwendung gestartet wurde, muss zuerst das Persönlichkeitsprofil erstellt werden. Hierzu klickt man auf den Punkt „Profil“. Ein Hinweis wird beim ersten Mal informieren, dass das Profil leer ist. Wenn man auf den Punkt „Profil neu“ klickt, erscheint ein Dialog, in dem die entsprechenden Werte ausgewählt oder in die entsprechenden Felder eintragen werden können. Die Größe muss hier in Zentimetern ohne Nachkommastellen angegeben werden. Der Suchradius wird in Kilometern, ebenfalls ohne Nachkommastellen, angegeben. Wenn die Auswahl zufrieden stellend ist, klickt man auf „Set“.

Profil ändern

Wenn das Profil später geändert werden soll, muss genauso wie oben beschrieben vorgegangen werden. Der Menüpunkt „Profil neu“ ist in diesem Fall automatisch durch „Profil ändern“ ersetzt worden. Die Vorgehensweise ist dieselbe wie beim Anlegen eines neuen Profils.

13.3.3 Partner finden

Wenn man auf den Punkt „Suche“ im Seitenmenü klickt, wird man mit einer Suchmaske konfrontiert, in welcher die Parameter des gewünschten Partners identifiziert werden können. Die Angaben können jeweils nur in Intervallen angegeben werden. Wenn die Auswahl „egal“ getroffen wird, wird der jeweilige Parameter nicht bei der Suche nach neuen Partnern berücksichtigt. Nachdem man seine Auswahl getroffen hat, klickt man auf „Suche“. Entweder wird jetzt eine Liste mit Benutzern angezeigt, die den Suchparametern entsprechen, oder eine Fehlermeldung, dass keine Partner gefunden werden konnten. Wenn eine Liste mit potenziellen Partnern angezeigt wird, können diese, indem auf das Icon rechts neben dem Namen des Benutzers geklickt wird, zur Buddyliste hinzugefügt werden. Wenn direkt auf den Namen des Benutzers geklickt wird, werden dessen Profildaten angezeigt. Um zur Auswahlliste zurückzukehren, kann die „Zurück“ Funktion des Browsers benutzt werden.

13.3.3.1 Partner lokalisieren

Um die Position eines Benutzers ausfindig zu machen, klickt man in irgendeiner Benutzerliste auf seinen Namen. Das kann die Buddyliste (siehe oben), die Ignoreliste (siehe 13.3.4 „Benutzer verwalten“) oder auch die Ergebnisliste der Partnersuche sein. Wenn man auf einen Benutzernamen klickt, werden dessen Profildaten angezeigt. Im Menü erscheint dann der zusätzliche Punkt „Lokalisieren“. Diese Funktion arbeitet nur dann, wenn die eigene Lokalisierung und die der anderen Person zugelassen sind. Informationen dazu finden sich in der Community Anleitung (siehe 13.1 „Community“). Wenn die Lokalisierung erfolgreich war, wird die Position des anderen Benutzers relativ zur eigenen angezeigt. Achtung, diese Anzeige wird nicht automatisch aktualisiert.

13.3.4 Benutzer verwalten

Für den Fall, dass ein Benutzer nicht mehr auf der Buddyliste gewünscht wird, kann er auch wieder gelöscht werden. Die Buddyliste bietet bei jedem Eintrag für einen Benutzer zwei Knöpfe an. Der Knopf unter der Überschrift „löschen“ entfernt den betreffenden Benutzer wieder von der Liste, ohne nachzufragen! Soll ein Benutzer nicht bloß gelöscht, sondern sollen in Zukunft auch keine Nachrichten mehr von ihm empfangen werden, so wählt man den Knopf unter „blocken“.



Abbildung 13.41: Die Optionen der Buddyliste.

Partner ignorieren

Eine Person, die mit dem „blocken“ Knopf von der Buddyliste entfernt wurde, wird in eine spezielle Ignoriert-Liste aufgenommen. Sämtliche Nachrichten von diesem Benutzer werden ignoriert, und er kann nicht durch eine erneute Suche wieder in die Buddyliste aufgenommen werden. Diese Liste kann man einsehen, indem bei der Ansicht des eigenen Profils (siehe 13.3.2 „Das eigene Profil“) auf den Menüpunkt „Ignorelist“ geklickt wird. In dieser Liste befinden sich alle Personen, die vorher mit einem Klick

auf den Blocken-Knopf von der Buddyliste gelöscht wurden. Ähnlich der Buddyliste gibt es auch hier wieder Zusatzoptionen. Neben dem Benutzernamen gibt es hier einen Knopf, um den entsprechenden Benutzer wieder zu entsperren und ihn wieder in die Buddyliste aufzunehmen.

13.3.5 Die Mailbox

Natürlich kann man mit den Personen, die mit der Dating Anwendung zu finden sind, auch kommunizieren. Das Seitenmenü bietet dazu den Punkt „Mailbox“. Am Anfang ist die Mailbox natürlich leer. Wenn in der eigenen Mailbox Nachrichten vorhanden sind, so werden diese als eine Liste angezeigt, ähnlich der Buddyliste.



Abbildung 13.42: Eine Nachricht lesen



Abbildung 13.43: Eine Beispielnachricht

Auch hier gibt es wieder die Kennung des Benutzers, der die Nachricht geschickt hat. Wie in den anderen Listen, so ruft ein Klick auf den Namen wieder das persönliche Profil dieser Person auf. Daneben finden sich Knöpfe zum Lesen und Löschen der Nachricht. Wenn man auf „löschen“ klickt, wird die Nachricht ohne Rückfrage gelöscht. Wenn man auf „lesen“ klickt, wird der Inhalt der Nachricht angezeigt. Um direkt auf die Nachricht zu antworten, kann man im Seitenmenü den Punkt „Nachricht beantworten“ wählen.

13.3.5.1 Eine Nachricht schreiben

Nachdem die Mailbox aufgerufen wurde, erscheint im Seitenmenü der Menüpunkt „Nachricht schreiben“. Nachrichten kann man nur an Benutzer schreiben, welche in der Buddyliste sind.



Abbildung 13.44: Eine Nachricht schreiben

Am oberen Rand findet sich eine Auswahlbox, um den Empfänger aus der Buddyliste auswählen zu können. Das Hauptfenster darunter ist für den eigentlichen Nachrichtentext. Wenn man einen Empfänger ausgewählt und den Text der Nachricht im Hauptfenster darunter eingegeben hat, klickt man auf den „Send“ Knopf. Die Nachricht wird jetzt verschickt. Falls man übrigens vom Empfänger ignoriert wird, erhält man keine Nachricht darüber.

13.3.5.2 Mailbox löschen

Statt alle Nachrichten einzeln zu löschen, kann auch die gesamte Mailbox auf einmal gelöscht werden. Hierzu klickt man im Mailbox Hauptmenü auf den Punkt „Mailbox leeren“. Anders als beim Löschen einzelner Nachrichten wird hier mit einer Sicherheitsabfrage überprüft, ob man wirklich sicher ist. Zum Löschen muss man natürlich mit „JA“ antworten.

13.3.6 Ein Beispiel: Wie man einen Partner findet.

Bevor Sie Ihre ersten Schritte im Datingbereich unternehmen, müssen Sie erst Daten angegeben, die Sie charakterisieren. Teile dieser Informationen haben Sie bereits beim ersten Einloggen angegeben, wie Ihr Geschlecht.

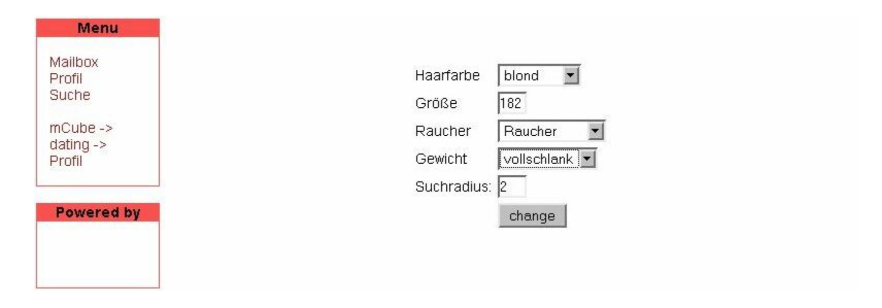


Abbildung 13.45: Was bin ich? Der Auswahlbildschirm für Ihre Profildaten.

Diese Daten sind unveränderlich. Die Daten, die Sie für Ihr Dating-Profil zusätzlich angeben, sind dagegen jederzeit veränderbar. Klicken Sie im linken Menü auf den Punkt „Profil“. Da dies Ihre erste Sitzung ist, werden Sie darüber informiert werden, dass Ihr Profil leer ist. Am linken Rand ist dafür der Menüpunkt „Profil neu“ hinzugekommen. Klicken Sie auf diesen.

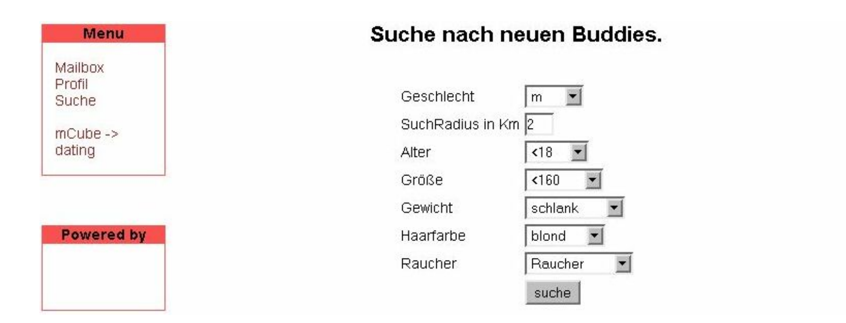


Abbildung 13.46: Partner gesucht.

Jeder dieser Punkte hat ein Auswahlmenu oder kann direkt mit Zahlen beschrieben werden. Geben Sie ihre Größe in Zentimetern an und den Suchradius in Kilometern um sich herum. (Keine Nachkommastellen!). Wählen Sie die übrigen Daten entsprechend ihres Aussehens aus. Wenn Sie fertig sind, klicken Sie auf den „Set“ Button am unteren Ende des Bildschirms, Sie werden danach über die erfolgreiche Speicherung ihrer Daten informiert werden. Die Menüpunkte am linken Rand sind wieder auf die Standardwerte zurückgesprungen, klicken Sie jetzt auf „Suche“.
Sie sehen jetzt wieder ein Auswahlmenü ähnlich dem Menü, welches Sie zur Bestimmung ihrer Profildaten vor sich hatten. Anders als dort werden Sie allerdings nur bei „Suchradius“ direkt einen Wert angeben können. Sie werden feststellen, dass hier auch eine Auswahlmöglichkeit für das Alter der gesuchten Person verfügbar ist und deren Geschlecht. Dieses Alter wird automatisch aus dem Geburtstag errechnet, welchen Sie beim ersten Einloggen in das System angegeben haben. Haben Sie die entsprechenden Charakteristika eines Wunschpartners ausgewählt, klicken Sie auf „Suche“.



Abbildung 13.47: Eine Auswahl an Personen.

Mit etwas Glück haben Sie nun eine Liste an Personen vor sich, welche dem gesuchten Profil entsprechen. Wenn Sie auf einen der Namen klicken, werden Sie sich die genauen Profildaten des Benutzers ansehen können. Dies ist übrigens immer möglich, wenn Sie eine Auswahl an Benutzern vor sich sehen, ein Klick auf den Benutzernamen zeigt Ihnen immer das entsprechende Profil an. Wenn Sie sich eine Person ausgesucht haben, klicken Sie auf das kleine Männchen rechts neben deren Namen. Sie werden mit einer Nachricht informiert werden, dass die Person erfolgreich hinzugefügt wurde. Kehren Sie mit einem Klick auf „Dating“, entweder in der oberen Menüleiste oder der Verlaufsanzeige im linken Menü zu ihrer Startseite zurück.
Sie werden nun eine Liste (die Buddyliste) mit dem von ihnen ausgewählten Benutzer



Abbildung 13.48: Glückwunsch, Sie haben ihren ersten Partner gefunden.

sehen. Glückwunsch, Sie haben nun einen potenziellen Partner gefunden, mit dem Sie Nachrichten austauschen und mit dem Sie sich treffen können.

Kapitel 14

Fazit

Wir haben die selbst gestellten Anforderungen (siehe Kapitel 7) weitestgehend erfüllen können. Nachteilig ausgewirkt hat sich hierbei, dass die Erfüllung der Anforderungen in starkem Maße von der Anbindung an Fremdsysteme abhing. Da wir Ortsinformationen, Shopsysteme und vieles anderes nicht selbst implementieren konnten, waren wir auf Anbindung angewiesen. Leider hat diese im Zeitrahmen der Projektgruppe nicht vollständig klappen können, was allerdings externe Gründe hat. Allerdings lässt dies viele Möglichkeiten für eine künftige Weiterführung der geleisteten Arbeit zu.

Die Arbeit und das Klima in der Projektgruppe waren sehr gut. Dies lässt sich natürlich auch darauf zurückführen, dass ein großer Teil der Teilnehmer sich schon vorher kannte. Zu größeren Differenzen innerhalb der Projektgruppe kam es nicht. Entscheidend hierfür war aber wohl auch, dass das Leistungsniveau der Teilnehmer ähnlich ist. Auf Grund der ähnlichen und guten Voraussetzungen gab es auch kaum Zeitverzug bei den Softwareinkrementen. Insgesamt hat die Zeitplanung also - vor allem auch wegen der Teilnehmer - gut funktioniert.

Die Zuteilung von Rollen war in den meisten Fällen sinnvoll. Entgegen der ursprünglichen Planung war es nicht nötig, die Rollen zu rotieren. Dies lag daran, dass die Rollen weder ausgenutzt noch fehlbesetzt wurden.

Das Erlernen von Projektarbeit ist als Ziel erreicht worden. Dies lässt sich auch daran festmachen, dass viele Kleingruppen eigenständig entwickelt haben und die Zusammensetzung zu einem Gesamtprodukt keine Probleme machte. Des Weiteren wurden die gestellten Anforderungen ohne großen Zeitverzug erreicht, und die Kompetenzen der Rollenzuteilung größtenteils eingehalten. Auch wenn ein Projekt in der Wirtschaft naturgemäß andere Voraussetzungen und Dimensionen hat, ist das Konzept der Projektarbeit allen Teilnehmern nähergebtracht worden.

Der Wirtschaftskontakt hat nicht in allen Belangen so funktioniert, wie wir uns das vorgestellt haben. Zwar war es gut möglich, Kontakt zu knüpfen, jedoch ist es schwer, eine Kooperation aufzubauen, von der beide Seiten profitieren. Letzten Endes reichte leider die Zeit der Projektgruppe für eine früchtetragende Kooperation nicht aus.

Kapitel 15

Ausblick

Wie im Fazit angedeutet, ergeben sich viele Möglichkeiten für eine Weiterführung der geleisteten Arbeit. Da unsere Architektur modern und flexibel ist, kann die Arbeit auch von anderen Personen weitergeführt werden. Notwendig ist hierbei die Anbindung an Fremdsysteme, die während der Laufzeit der Projektgruppe nicht möglich war. Möglich ist eine Erweiterung der Funktionalität sowohl durch Ausbau der bestehenden Komponenten als auch durch Einbindung neuer Komponenten. Bei den bestehenden Komponenten sind in der Anforderungsanalyse (siehe Kapitel 7) einige KANN-Anforderungen übriggeblieben, die das System sinnvoll erweitern. Die flexible Architektur erlaubt auch die einfache Anpassung an weitere Endgeräte. An vielen Stellen können noch ortsbasierte Dienste eingebaut werden und zu einem Mehrwert des Produkts führen. Es bleibt zu hoffen, dass das Projekt in irgendeiner Form weitergeführt wird.

Literaturverzeichnis

[Apa] Apache Project. Xalan2-java. http://xml.apache.org/xalan-j/index.html.

[Bea03] Bea. Weblogic server 7.0. http://e-docs.bea.com/wls/docs70/index.html, 2003.

[Bec99] Kent Beck. *Extreme Programming Explained*. Addison-Wesley Professional, 1999.

[BPSM00] Tim Bray, Jean Paoli, C.M. Sperberg, and Eve Maler. Extensible markup language (xml) 1.0 (second edition). http://www.w3org/TR/2000/REC-xml-20001006, 2000. Zitiert: 18.03.2003.

[Bro97] V. Broady. Trading Card Game takes net by storm. http://www.zdnet.com/anchordesk/story/story_1299.html, 1997. Abruf: 22.02.2003.

[BT75] V. R. Basili and A. J. Turner. Iterative Enhancement: A Practical Technique for Software Development. In *IEEE Trans. Software Engineering*, volume 1(4), pages 390–396. 1975.

[Cla99] James Clark. Xsl transformations (xslt) version 1.0. http://www.w3org/TR/1999/REC-xslt-19991116, 1999. Zitiert: 18.03.2003.

[Cvs] The CvsGui team. Cvsgui. http://www.wincvs.org/.

[FF03] Martin Fowler and Matthew Foemmel. Continuous integration. http://www.martinfowler.com/articles/continuousIntegration.html, März 2003.

[IBM98] IBM Redbooks. *Understanding LDAP*. IBM Corporation, 1998.

[Int01] InterMedia Interactive Solutions Inc. Driving business profitability through online community. http://www.intermediainc.com/flash/clienttestpdf/perspectives/CommunityDrivsProfitability.pdf, 2001. Abruf: 22.02.2003.

[KAL$^+$01] H. Kaaranen, A. Ahtiainen, L. Laitinen, S. Naghian, and V. Niemi. *UMTS Network : Architecture, Mobility and Services*. John Wiley and Sons Ltd., 2001.

[LF02] Johannes Link and Peter Fröhlich. *Unit Tests mit Java*. dpunkt.verlag, Januar 2002.

[LR02] Bennet P. Lientz and Kathryn P. Rea. *Project Management for the 21st Century*. Academic Press, 2002.

[Lyc02] Lycos Europe. BMG und Lycos Europe pushen Online-Werbung. http://www.lycos.de/ content/service/cc/press-center/archive/20020819a.html, 2002. Abruf: 22.02.2003.

[Mic03] Microsoft. Active directoy. http://www.microsoft.com/windows2000/ technologies/directory/ad/default.asp, 2003.

[MM00] Jack R. Meredith and Samuel J. Mantel. *Project Management, A Managerial Approach*. John Wiley & Sons, 2000.

[Net03] Netscape. Netscape directory server. http://enterprise.netscape.com/ products/identsvcs/directory.html, 2003.

[Nor01] Northstream AB. Location Based Services. http://www.northstream.se/ download/LocationBasedServices.pdf, 2001. Abruf: 22.02.2003.

[Obj] Object Mentor Incorporated. Junit.org. http://www.junit.org/index.htm.

[Ses99] Govind Seshandri. Understanding java server pages model 2 architecture - exploring the mvc design pattern. *JavaWorld*, Dezember 1999.

[Sha01] Bill Shannon. *Java 2 Platform Enterprise Edition Specification, v1.3*. Sun Microsystems, Inc., Juli 2001.

[Sun] Sun Microsystems Inc. Javadoc tool home page. http://java.sun.com/ j2se/javadoc/.

[Sun03a] Sun. Java message service api. http://java.sun.com/products/jms/, 2003.

[Sun03b] Sun Microsystems Inc. Enterprise javabeanstm technology. http://java.sun.com/products/ejb/, 2003.

[Sun03c] Sun Microsystems Inc. *Java 2 Platform, Micro Edition (J2ME)*. Sun Microsystems, Inc., März 2003.

[TB02] Jesse Tilly and Eric M. Burke. *Ant: The Definitive Guide*. O'Reilly & Associates, May 2002.

/99/ T. Bühren, M. Cakir, E. Can, A. Dombrowski, G. Geist, V. Gruhn, M. Gürgrn, S. Handschumacher, M. Heller, C. Lüer, D. Peters, G. Vollmer, U. Wellen, J. von Werne
Endbericht der Projektgruppe eCCo (PG 315)
Electronic Commerce in der Versicherungsbranche
Beispielhafte Unterstützung verteilter Geschäftsprozesse
Februar 1999

/100/ A. Fronk, J. Pleumann,
Der DoDL-Compiler
August 1999

/101/ K. Alfert, E.-E. Doberkat, C. Kopka
Towards Constructing a Flexible Multimedia Environment for Teaching the History of Art
September 1999

/102/ E.-E. Doberkat
An Note on a Categorial Semantics for ER-Models
November 1999

/103/ Christoph Begall, Matthias Dorka, Adil Kassabi, Wilhelm Leibel, Sebastian Linz, Sascha Lüdecke, Andreas Schröder, Jens Schröder, Sebastian Schütte, Thomas Sparenberg, Christian Stücke, Martin Uebing, Klaus Alfert, Alexander Fronk, Ernst-Erich Doberkat
Abschlußbericht der Projektgruppe PG-HEU (326)
Oktober 1999

/104/ Corina Kopka
Ein Vorgehensmodell für die Entwicklung multimedialer Lernsysteme
März 2000

/105/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Zwischenbericht der Projektgruppe IPSI
April 2000

/106/ Ernst-Erich Doberkat
Die Hofzwerge — Ein kurzes Tutorium zur objektorientierten Modellierung
September 2000

/107/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
November 2000

/108/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Endbericht der Projektgruppe IPSI
Februar 2001

/109/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
Februar 2001

/110/ Eugenio G. Omodeo, Ernst-Erich Doberkat
Algebraic semantics of ER-models from the standpoint of map calculus.
Part I: Static view
März 2001

/111/ Ernst-Erich Doberkat
An Architecture for a System of Mobile Agents
März 2001

/112/ Corina Kopka, Ursula Wellen
Development of a Software Production Process Model for Multimedia CAL Systems by Applying Process Landscaping
April 2001

/113/ Ernst-Erich Doberkat
The Converse of a Probabilistic Relation
Oktober 2002

/114/ Ernst-Erich Doberkat, Eugenio G. Omodeo
Algebraic semantics of ER-models in the context of the calculus of relations.
Part II: Dynamic view
Juli 2001

/115/ Volker Gruhn, Lothar Schöpe (Eds.)
Unterstützung von verteilten Softwareentwicklungsprozessen durch integrierte Planungs-, Workflow- und Groupware-Ansätze
September 2001

/116/ Ernst-Erich Doberkat
The Demonic Product of Probabilistic Relations
September 2001

/117/ Klaus Alfert, Alexander Fronk, Frank Engelen
Experiences in 3-Dimensional Visualization of Java Class Relations
September 2001

/118/ Ernst-Erich Doberkat
The Hierarchical Refinement of Probabilistic Relations
November 2001

/119/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer, Ingo Röpling,
Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Intermediate Report
November 2001

/120/ Volker Gruhn, Ursula Wellen
Autonomies in a Software Process Landscape
Januar 2002

/121/ Ernst-Erich Doberkat, Gregor Engels (Hrsg.)
Ergebnisbericht des Jahres 2001
des Projektes "MuSofT – Multimedia in der SoftwareTechnik"
Februrar 2002

/122/ Ernst-Erich Doberkat, Gregor Engels, Jan Hendrik Hausmann, Mark Lohmann, Christof Veltmann
Anforderungen an eine eLearning-Plattform — Innovation und Integration —
April 2002

/123/ Ernst-Erich Doberkat
Pipes and Filters: Modelling a Software Architecture Through Relations
Juni 2002

/124/ Volker Gruhn, Lothar Schöpe
Integration von Legacy-Systemen mit Eletronic Commerce Anwendungen
Juni 2002

/125/ Ernst-Erich Doberkat
A Remark on A. Edalat's Paper *Semi-Pullbacks and Bisimulations in Categories of Markov-Processes*
Juli 2002

/126/ Alexander Fronk
Towards the algebraic analysis of hyperlink structures
August 2002

/127/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer
Ingo Röpling, Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Final Report
August 2002

/128/ Timo Albert, Zahir Amiri, Dino Hasanbegovic, Narcisse Kemogne Kamdem,
Christian Kotthoff, Dennis Müller, Matthias Niggemeier, Andre Pavlenko, Stefan Pinschke,
Alireza Salemi, Bastian Schlich, Alexander Schmitz
Volker Gruhn, Lothar Schöpe, Ursula Wellen
Zwischenbericht der Projektgruppe Com42Bill (PG 411)
September 2002

/129/ Alexander Fronk
An Approach to Algebraic Semantics of Object-Oriented Languages
Oktober 2002

/130/ Ernst-Erich Doberkat
Semi-Pullbacks and Bisimulations in Categories of Stochastic Relations
November 2002

/131 Yalda Ariana, Oliver Effner, Marcel Gleis, Martin Krzysiak,
Jens Lauert, Thomas Louis, Carsten Röttgers, Kai Schwaighofer,
Martin Testrot, Uwe Ulrich, Xingguang Yuan
Prof. Dr. Volker Gruhn, Sami Beydeda
Endbericht der PG nightshift:
Dokumentation der verteilten Geschäftsprozesse im FBI und Umsetzung von Teilen dieser Prozesse im Rahmen eines FBI-Intranets basierend auf WAP- und Java-Technologie
Februar 2003

/132/ Ernst-Erich Doberkat, Eugenio G. Omodeo
ER Modelling from First Relational Principles
Februar 2003

/133/ Klaus Alfert, Ernst-Erich Doberkat, Gregor Engels (Hrsg.)
Ergebnisbericht des Jahres 2002 des Projektes "MuSofT – Multimedia in der SoftwareTechnik"
März 2003

/134/ Ernst-Erich Doberkat
Tracing Relations Probabilistically
März 2003

/135/ Timo Albert, Zahir Amiri, Dino Hasanbegovic, Narcisse Kemogne Kamdem,
Christian Kotthoff, Dennis Müller, Matthias Niggemeier,
Andre Pavlenko, Alireza Salemi,Bastian Schlich, Alexander Schmitz,
Volker Gruhn, Lothar Schöpe, Ursula Wellen
Endbericht der Projektgruppe Com42Bill (PG 411)
März 2003

/136/ Klaus Alfert
Vitruv: Specifying Temporal Aspects of Multimedia Presentations —
A Transformational Approach based on Intervals
April 2003

/137/ Klaus Alfert, Jörg Pleumann, Jens Schröder
A Framework for Lightweight Object-Oriented Design Tools
April 2003

/138/ K. Alfert, A. Fronk , Ch. Veltmann (Hrsg.)
Stefan Borggraefe, Leonore Brinker, Evgenij Golkov, Rafael Hosenberg, Bastian Krol, Daniel Mölle,
Markus Niehammer, Ulf Schellbach, Oliver Szymanski, Tobias Wolf, Yue Zhang
Endbericht der Projektgruppe 415: Konzeption und Implementierung eines digitalen und hypermedialen Automobilcockpits (HyCop)
Mai 2003

/139/ Volker Gruhn, Malte Hülder, Sami Beydeda (Hrsg.)
Endbericht der Projektgruppe 409: Entwicklung von ortsbasierten Diensten für UMTS-Mobilfunkgeräte (mCube)
Mai 2003